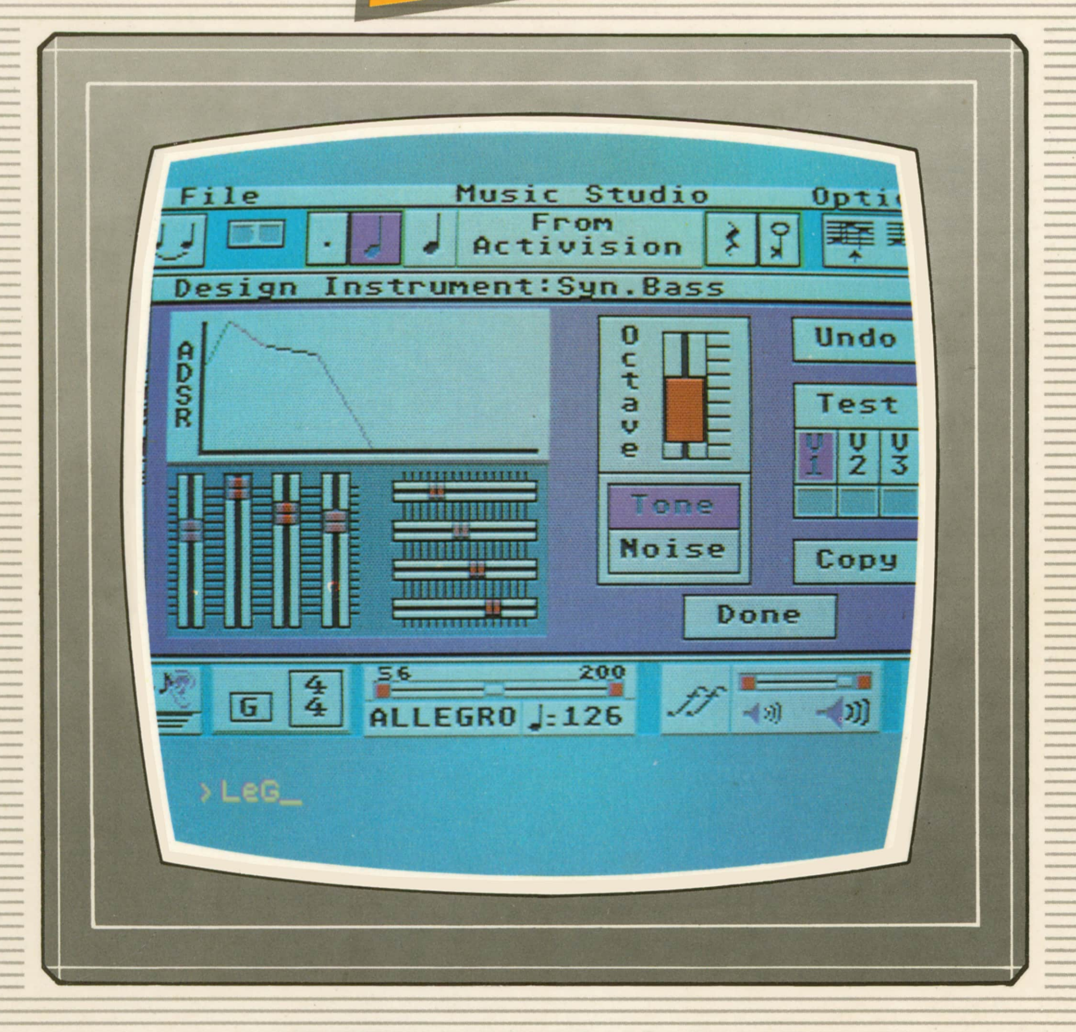
▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación 33

PASO A PASO



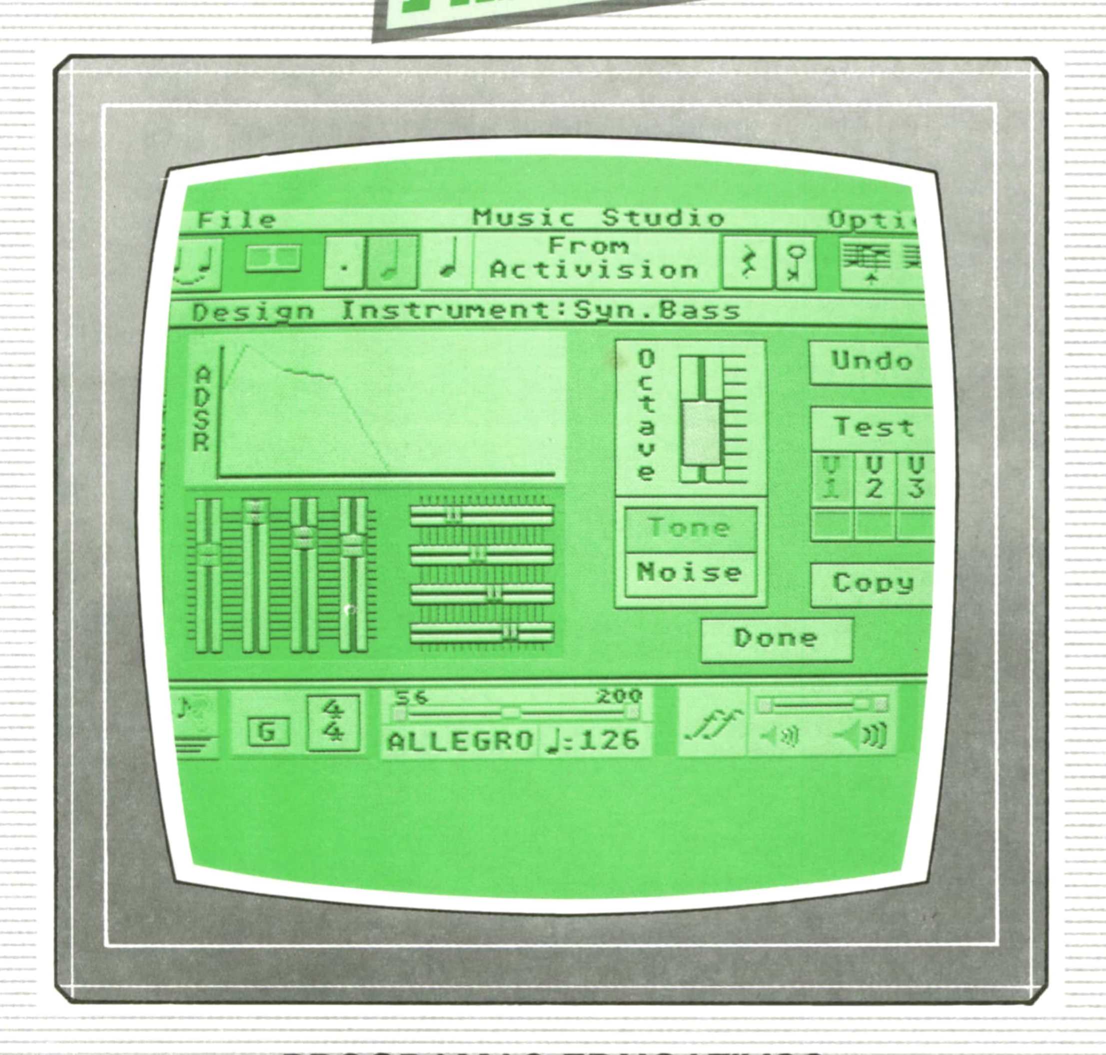
PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 33

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.

Diseño y maquetación:
BRAVO-LOFISH.

Fotografía:
EQUIPO GALATA.

Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas. Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Jorge Thomas, Técnico en Informática. OTROS LENGUAJES (ADA): Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación.

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 455 09 99. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: **84-7688-184-3**
ISBN de la obra: 84-7688-068-7

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.

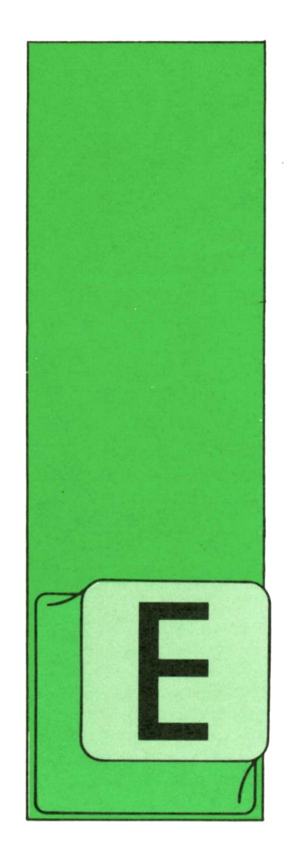
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 259 73 31. 28020 Madrid.

Febrero, 1988
P.V.P. Canarias: 335,–.

INDICE

BASIC

MATRICES III



Matrices multidimensionales

EL procedimiento para trabajar con matrices de más de dos dimensiones es similar al visto hasta el momento. Por lo general, por cada dimensión más tendremos que utilizar un bucle anidado más, ya que las variables tendrán un subíndice más.

De todas formas, no es muy corriente utilizar matrices de más de dos dimensiones, en particular si son alfanuméricas.

En cualquier caso, vamos a ver un ejemplo de utilización de una matriz numérica tridimensional. El programa 3 simula la taquilla automática de venta de entradas de un hipotético cine que cuenta con dos pisos, cada uno de los cuales dispone de 16 filas con 20 butacas por fila.

```
10 REM **********************
20 REM *  TAQUILLA DE CINE  *
30 REM **********************
40 CLS
50 LET T=0
60 DIM C(15,20,2)
70 INPUT "FILA (1-15)";F
80 IF F<1 OR F>15 THEN GOTO 70
90 INPUT "BUTACA (1-20)";B
100 IF B<1 OR B>20 THEN GOTO 90
110 INPUT "PISO (1-2)";P
120 IF P<>1 AND P<>2 THEN GOTO 110
130 CLS
140 IF C(F,B,P)=1 THEN PRINT "OCUPADA":PRINT :GOTO 190
150 LET C(F,B,P)=1
160 LET T=T+1
170 PRINT "AQUI TIENE SU ENTRADA"
180 PRINT
190 IF T<F*B*P THEN GOTO 70
200 CLS
210 PRINT :PRINT
220 PRINT "AGOTADAS LAS LOCALIDADES"
```

En este caso la matriz sólo almacena ceros y unos. Si una posición de la matriz contiene un cero, significa que la butaca que representa está vacía, mientras que si contiene un uno la butaca está ocupada.

Características comunes en matrices de cualquier dimensión

1. Si el índice de la variable al dimensionar no es un número entero, el ordenador tomará la parte entera:

DIM A(16.5) equivale a DIM A(16) y
DIM A(15.1, 17.7) equivale a DIM A(15,17)

2. En la instrucción DIM se puede escribir entre paréntesis un número, una expresión o incluso el nombre de otra variable a la que previamente se le haya asignado contenido:

Serían, pues, correctas líneas del tipo:

DIM A(6 * 5 + 1,7) o DIM A(M,N)

3. El nombre genérico matriz sigue las mismas normas de formación que el nombre de cualquier variable, excepto en el SPECTRUM, que sólo puede tener una letra.

4. No tiene sentido escribir como subíndice un número negativo. Si lo hace obtendrá un mensaje de error.

5. Dentro de un programa sólo podemos referirnos en un momento dado a un componente de la matriz (A(I), A(I,J), etcétera), y no a la matriz completa. De hecho, en el mismo programa se podrían utilizar las variables A y A(I).

6. Excepto en el SPECTRUM, no es necesario dimensionar si la matriz va a constar de menos de 11 elementos (si es de una dimensión), menos de 10 × 10 elementos (si es de dos), etc. Sin embargo, puede resultar útil hacerlo para que el ordenador no reserve memoria en exceso.

7. Es posible dimensionar, por ejemplo, A(20) y trabajar sólo con A(1), A(2), A(3) y A(4); lo único que ocurre es que parte de memoria reservada no se utiliza. Pero daría error intentar trabajar con A(21), A(22), etc.

8. Al igual que en el caso de las variables simples, el ordenador inicializa las matrices, asignando el valor 0 a todas sus posiciones si se trata de una matriz numérica y la cadena nula si es alfanumérica; por tanto, si no se asignan valores específicos a algunas posiciones de una matriz, su valor será cero (matriz numérica) o cadena nula (matriz alfanumérica). Incluso en el SPECTRUM, en el que era necesario inicializar todas las variables, el hecho de dimensionar implica poner a cero todos los elementos de la matriz si es numérica y a cadena nula si es alfanumérica.

9. No se puede dimensionar la misma matriz dos veces en el mismo programa.

Unicamente en el SPECTRUM no daría error por la redimensión de una matriz.



Subrutinas

Al ir profundizando en el estudio del BASIC vamos teniendo más clara la idea de que cada vez podemos diseñar programas más complejos y no va siendo suficiente tener como único criterio que «el programa funcione».

Tan importante como que el programa funcione es que tenga una lógica clara para facilitar la localización de errores, modificaciones, etc.

Para ello es necesario estructurar los programas lo más posible. Esto se puede conseguir con la utilización de subrutinas.

Una subrutina, también llamada subprograma o ruina, no es más que un conjunto de líneas de programa que realizan una determinada tarea. En muchos casos dicha tarea será necesaria en distintas partes de un programa. Las subrutinas sólo se escriben una vez en un programa, pero se pueden ejecutar tantas veces como sea necesario y en los lugares del programa que haga falta.

El empleo de subrutinas permite, en muchos casos, acortar el tamaño de los programas, a la vez que aumenta la efectividad de los mismos.

Los programas BASIC se jerarquizan con el empleo de subrutinas. El *programa principal* constituye la jerarquía más alta y está formado por una serie de instrucciones situadas al principio del programa. El programa principal controla el funcionamiento general del programa y pide la ejecución de las subrutinas cuando es necesario.

La instrucción que produce la «llamada» a una subrutina es GOSUB, cuyo formato es el siguiente:

GOSUB <número de línea>

Una instrucción GOSUB dentro de un programa obliga a que se transfiera el control a la línea especificada a conti-

nuación del GOSUB. Sin embargo, esto no es equivalente a la instrucción GOTO, ya que existe otra sentencia, RETURN que indica el final de la subrutina, de modo que cuando la ejecución de la subrutina llega a un RETURN, el control vuelve de nuevo al programa principal, a la instrucción siguiente al GOSUB.

En la figura 1 podemos ver este efecto de «ida y vuelta».

En la figura 2 vemos el esqueleto típico de un programa bien estructurado.

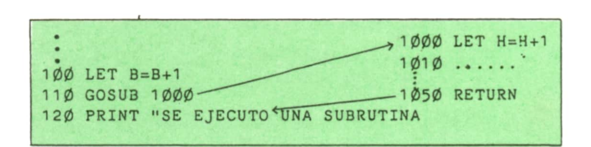




Fig. 1. Efecto de «ida y vuelta» de las subrutinas.

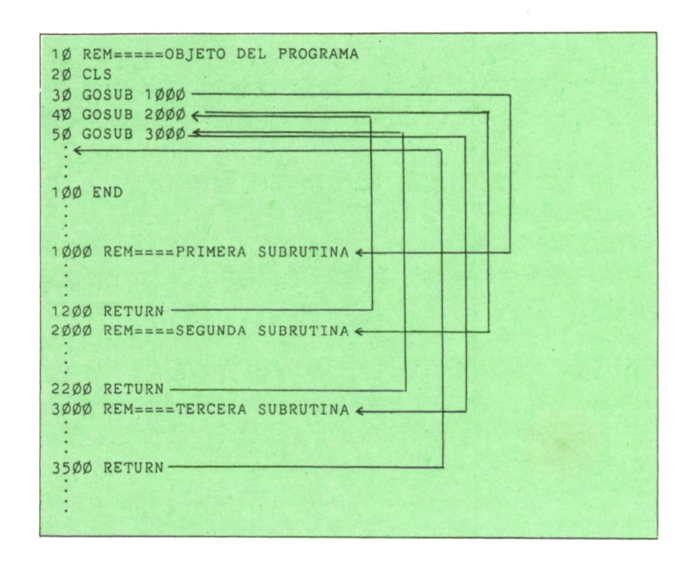




Fig. 2. Esqueleto básico de un programa estructurado con subrutinas.

Las líneas desde la 10 hasta la 100 constituirían el programa principal desde el cual se controlan tres subrutinas.

El empleo de subrutinas evita, como ya hemos dicho, repetir ciertas tareas cada vez que son necesarias.

Supongamos que vamos a diseñar un programa que frecuentemente debe solicitar un dato numérico y no se debe admitir números negativos. El esqueleto básico sería el representado en la figura 3.

También es muy frecuente que desde dentro de una subrutina se llame a otra.

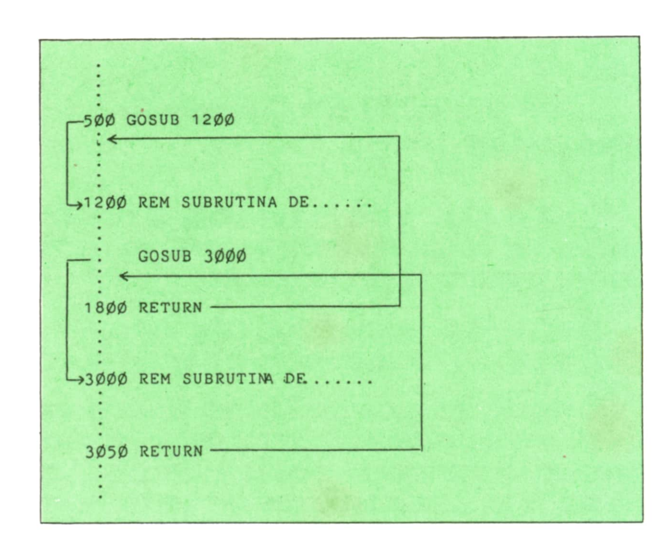




Fig. 3. Esqueleto básico de un programa que llama varias veces a una misma subrutina.

Este caso se muestra en el esqueleto de la figura 4.

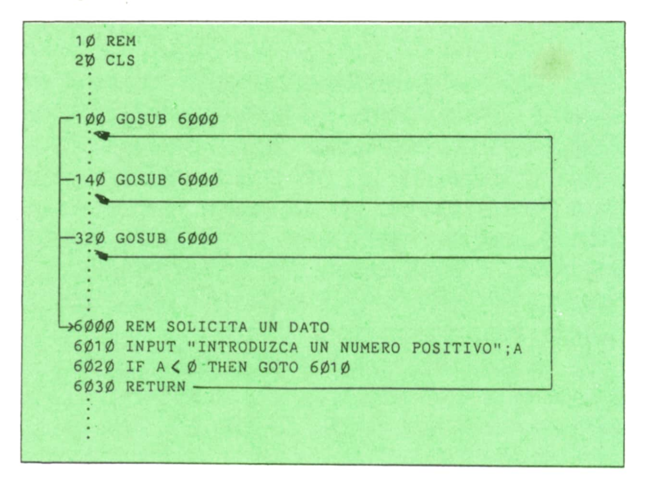




Fig. 4. Esqueleto básico de un programa en el que una subrutina llama a otra.

Por otra parte, hay que señalar que sólo se puede entrar en una subrutina con la instrucción GOSUB, ya que de lo contrario aparecerá en pantalla un mensaje de error: RETURN WITHOUT GOSUB. Esto se debe a que el ordenador encontraría una instrucción RETURN y no sabría a qué parte del programa volver, puesto que no habría pasado por el GOSUB correspondiente.

Es muy frecuente la utilización de subrutinas en programas que presentan un MENU DE OPCIONES, de modo que, según la opción seleccionada, se ejecutará una subrutina u otra.

El programa 1 contiene todo lo explica-

do hasta el momento: varias llamadas a una misma subrutina, subrutinas que llaman a otras subrutinas y menú de opciones.

Su objetivo es poder elegir sumas, restas o multiplicaciones. Los operandos son generados al azar por el ordenador. El usuario debe acertar los resultados.

```
10 REM *******************************
20 REM *  OPERACIONES  ARITMETICAS  *
30 REM *******************************
40 CLS
50 PRINT TAB(17);"OPCIONES"
60 PRINT TAB(17);"________"
70 PRINT :PRINT :PRINT
80 PRINT TAB(17);"1. SUMAS"
90 PRINT
100 PRINT TAB(17);"2. RESTAS"
110 PRINT
120 PRINT TAB(17);"3. MULTIPLICACIONES"
130 PRINT
140 PRINT TAB(17);"4. TERMINAR"
150 PRINT :PRINT :PRINT
160 PRINT "PULSA LA OPCION DESEADA"
170 LET R$=INKEY$:IF R$="" THEN GOTO 170
180 IF ASC(R$)<49 OR ASC(R$)>52 THEN GOTO 170
190 CLS
200 IF R$="1" THEN GOSUB 1000
210 IF R$="2" THEN GOSUB 2000
220 IF R$="3" THEN GOSUB 3000
230 IF R$="4" THEN END
240 GOTO 40
1000 REM *  SUMAS  *
1010 GOSUB 4000
1020 PRINT A;"+";B;"=";
1030 INPUT R
1040 PRINT :PRINT
1050 IF R=A+B THEN PRINT "CORRECTO":GOTO 1070
1060 PRINT "INCORRECTO. LA RESPUESTA ES ";A+B
1070 GOSUB 5000
1080 RETURN
2000 REM *  RESTAS  *
2010 GOSUB 4000
2020 PRINT A;"-";B;"=";
2030 INPUT R
2040 PRINT :PRINT
2050 IF R=A-B THEN PRINT "CORRECTO":GOTO 2070
2060 PRINT "INCORRECTO. LA RESPUESTA ES ";A-B
2070 GOSUB 5000
2080 RETURN
3000 REM *  MULTIPLICACIONES  *
3010 GOSUB 4000
3020 PRINT A;"*";B;"=";
3030 INPUT R
3040 PRINT :PRINT
3050 IF R=A*B THEN PRINT "CORRECTO":GOTO 3070
3060 PRINT "INCORRECTO. LA RESPUESTA ES ";A*B
3070 GOSUB 5000
3080 RETURN
4000 REM *  GENERACION ALEATORIA DE NUMEROS  *
4010 LET A=INT(RND(1)*10000)
4020 LET B=INT(RND(1)*10000)
4030 RETURN
5000 REM *  PAUSA ANTES DE VOLVER AL MENU  *
5010 PRINT :PRINT :PRINT
5020 PRINT "PULSA UNA TECLA PARA CONTINUAR"
5030 LET T$=INKEY$:IF T$=""THEN GOTO 5030
5040 RETURN
```

En las figuras 5 y 6 podemos ver la presentación en pantalla del menú de opciones así como el aspecto de la pantalla durante la ejecución de una de las subrutinas principales.

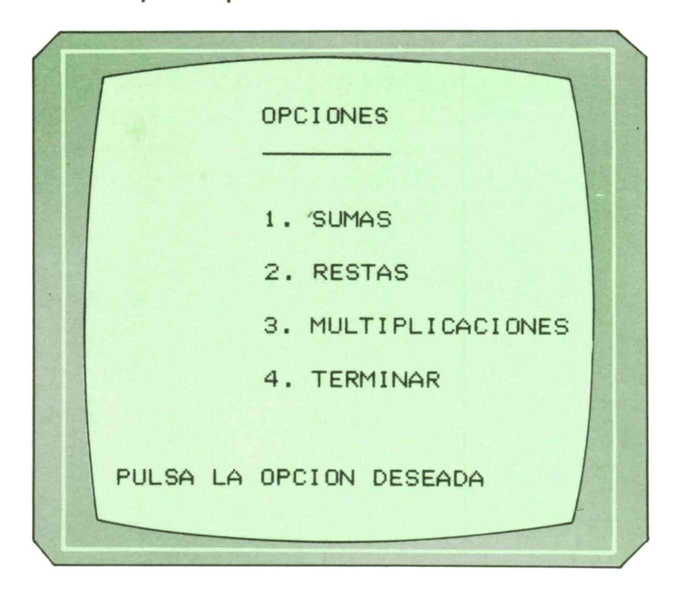



Fig. 5. Presentación en pantalla del menú de opciones del programa 1.




Fig. 6. Presentación en pantalla de una de las opciones del programa 1.

Sin embargo, en el caso de los menús, donde la opción siempre es múltiple, resulta más adecuado utilizar la instrucción ON-GOSUB, que proporciona otra forma de direccionamiento hacia las distintas subrutinas. Su formato es el siguiente:

ON <expresión> GOSUB n.° línea, n.° línea...

Entre las palabras ON y GOSUB se escribe una variable numérica o una expresión aritmética que al evaluarse decidirá la subrutina a la que debe dirigirse la ejecución del programa.

Tras el GOSUB se escribe una lista de números de líneas separados por comas que corresponden a otras tantas subrutinas.

La subrutina que se ejecutará es aquella cuya posición en la lista situada tras el GOSUB coincide con el valor de la variable o expresión.

Podemos ver un ejemplo en la figura 7.

```
100 ON I GOSUB 400,500,
    600
```

VALOR DE I	RUTINA EJECUTADA
1	400
2	500
3	600


Fig. 7. Relación entre el valor de la variable y la rutina ejecutada en la instrucción ON-GOSUB.

Hay que tener en cuenta que si la variable toma un valor superior al número de líneas que componen la lista no se ejecutará ninguna subrutina y la ejecución continuará por la línea siguiente al ON-GOSUB. Sin embargo, si la variable toma un valor negativo se producirá un error.

Por otra parte, hay que advertir que el SPECTRUM no dispone de la instrucción ON-GOSUB. Sin embargo, podemos conseguir un efecto similar con el siguiente formato para GOSUB:

GOSUB <expresión>

De este modo no es necesario indicar tras el GOSUB un número de línea concreto, sino que podemos especificar una variable o una expresión aritmética que al evaluarse determinará el número de línea correspondiente a la subrutina.

Por ejemplo, la instrucción ON-GOSUB que vimos en la figura 7 tendría el siguiente formato en el SPECTRUM:

GOSUB (I+3)*100

Para finalizar, el programa 2 permite la resolución de ecuaciones de segundo grado. El programa está estructurado en tres subrutinas dependiendo del tipo de

```
10 REM **********************************
20 REM *  ECUACIONES DE SEGUNDO GRADO   *
30 REM **********************************
40 CLS
50 INPUT "COEFICIENTES A,B Y C DE LA ECUACION";A,B,C
60 CLS
70 LET D=B^2-4*A*C
80 LET S=SGN(D)+2
90 ON S GOSUB 200,300,400
100 PRINT :PRINT :PRINT :PRINT
110 PRINT "¿QUIERES RESOLVER OTRA ECUACION? (S/N)"
120 LET R$=INKEY$:IF R$="" THEN GOTO 120
130 IF R$="S" OR R$="s" THEN GOTO 40
140 END
200 REM *  RAICES IMAGINARIAS  *
210 PRINT :PRINT
220 PRINT "RAICES IMAGINARIAS"
230 PRINT :PRINT
240 LET R=-B/(2*A)
250 LET I=ABS(SQR(ABS(D))/(2*A))
260 PRINT "X1 = ";R;"+";I;"i"
270 PRINT
280 PRINT "X2 = ";R;"-";I;"i"
290 RETURN
300 REM *  RAIZ REAL DOBLE  *
310 PRINT :PRINT
320 PRINT "RAIZ REAL DOBLE"
330 PRINT :PRINT
340 LET R=-B/(2*A)
350 PRINT "X = ";R
360 RETURN
400 REM *  RAICES REALES DISTINTAS  *
410 PRINT :PRINT
420 PRINT "RAICES REALES DISTINTAS"
430 PRINT :PRINT
440 LET R1=(-B+SQR(D))/(2*A)
450 LET R2=(-B-SQR(D))/(2*A)
460 PRINT "X1 = ";R1
470 PRINT
480 PRINT "X2 = ";R2
490 RETURN
```

soluciones de la ecuación: imaginarias, reales distintas o reales dobles.

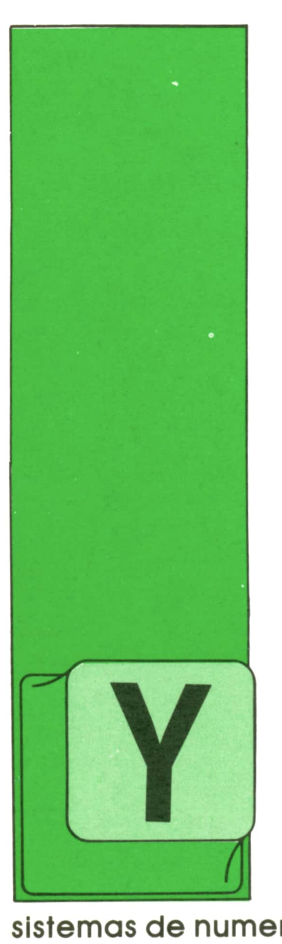
Sólo recordar que en los dos programas vistos habrá que sustituir las asignaciones tipo LET R$=INKEY$ por GET R$ y los CLS por PRINT CHR$(147) si trabajamos con un COMMODORE.

En el SPECTRUM habrá que cambiar la línea 90 del programa 2 por:

```
90 GOSUB (S+1)*100
```

MAQUINA Z-80

SPECTRUM, AMSTRAD, MSX





Ensamblador

Y A ha quedado claro que el lenguaje máquina es físicamente una «ristra» de unos y ceros, números en binario, que normalmente se representan con cifras en otros sistemas de numeración más manejables (decimal, hexadecimal, octal, etc.). Programar en máquina utilizando números exclusivamente, es un trabajo de titanes, y además inútil, gracias a los programas ensambladores.

Los ensambladores, además de traducir los códigos nemónicos a sus correspondientes números, permiten la utilización de etiquetas y otras utilidades que facilitan enormemente el desarrollo de programas en lenguaje máquina (en este caso lenguaje ensamblador).

Para cada microprocesador diferente existen diversos programas ensambladores. En este caso nos referiremos al programa llamado MACRO-80, un ensamblador que está disponible para la mayoría de los ordenadores personales y que permite ensamblar códigos de Z-80, 8080 e incluso 6402. De todas formas, la nomenclatura de los ensambladores es muy similar.

Expliquemos brevemente cuál es el proceso para realizar un programa ejecutable en lenguaje máquina.

En primer lugar, se debe realizar la codificación en lenguaje ensamblador del algoritmo a utilizar en el programa y escribir este programa en un fichero de texto mediante un editor de texto cualquiera. Este fichero puede archivarse en el dispositivo periférico a utilizar. (El más aconsejable por su velocidad es el disco, pero son posibles otros, como la tinta de cassette, etc.

Una vez realizado el programa en texto (se le suele llamar programa fuente), cargaremos y ejecutaremos el programa ensamblador que realizará la traducción del archivo de texto a lenguaje máquina (llamado también programa objeto) y que será almacenado en el mismo dispositivo del que leyó el programa fuente. El programa obtenido de esta forma no es realmente ejecutable, ya que el macroensamblador Macro-80 realiza un ensamblado intermedio de código relocalizable. Para que el código sea ejecutable debemos utilizar un segundo programa, editor de enlaces simbólicos (Linker) con el que obtendremos un programa ya ejecutable. La nomenclatura de este segundo programa es bastante complicada, por lo que es aconsejable un estudio detallado del manual.

La ventaja de este método es el ensamblado por partes: se hacen pequeñas rutinas que se ensamblan por separado y luego se unen con el «LINKER» confeccionando así programas grandes de forma modular.

Expliquemos brevemente la nomenclatura del macroensamblador: la forma general de una línea de programa es:

```
ETIQUETA:   LD    A,1000H;CARGA ACUM.
```

Etiqueta de Dirección Código de Operación Argumentos Comentar.

todas las partes de la línea son opcionales.

La etiqueta es un punto de referencia utilizable para direccionar la instrucción en el módulo donde se define la etiqueta. Se pueden definir etiquetas como PUBLIC, con lo que son utilizables por otros módulos, o como EXTERNAL, permitiendo definirse fuera del programa.

El ensamblador tiene varios modos de funcionamiento dependiendo de si se desea que el código objeto sea relocalizable o no.

Por supuesto, los argumentos pueden ser expresiones aritmético-lógicas, en las que se incluyen variables; su contenido puede ser 8 ó 16 bits, dependiendo del caso.

Al inicializar en ensamblador, éste está en el sistema de ensamblado para 8080, por lo que debemos usar la directiva que cambia el sistema al del Z-80. Esta es de la forma:

.z80

Con ésta, el código fuente será tratado como perteneciente al Z-80. En caso contrario el ensamblador trataría el programa como si fuese de 8080, y al poseer una nomenclatura distinta, detectaría numerosos errores. La asignación de etiquetas que no se realiza de forma relativa (sólo en el caso de direcciones) se puede asignar con la sentencia:

<NOMBRE> EQU <EXPRESION >

El módulo en ensamblador debe finalizarse con la sentencia END.

Debemos diferenciar entre sentencias de programa y directivas. Las directivas son órdenes que afectarán al ensamblado cambiando la forma de realizarse el mismo, pero que no formarán parte del código objeto. Suelen ir precedidas de un punto ".", por lo que son fácilmente diferenciables.

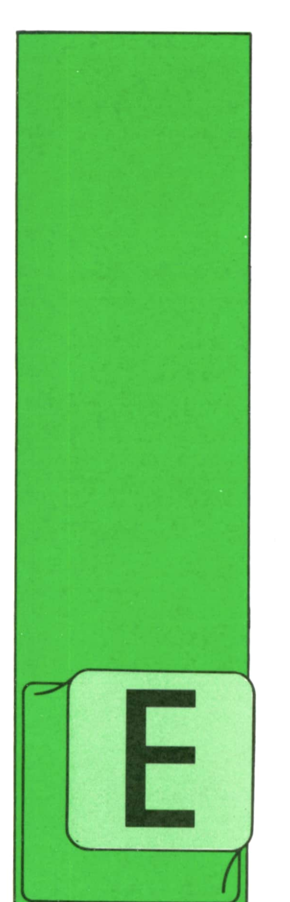
Dependiendo de la calidad del macroensamblador que utilicemos, habrá un mayor número de directivas que ampliarán el número de opciones permitiendo una mayor versatilidad.

Una de las opciones más sofisticadas es la de ensamblado condicional. Esta permite que dependiendo del valor que tenga una determinada expresión lógica en la que puede haber variables del programa, se ensamble o no un grupo de instrucciones.

Esto permite que insertemos o no código, dependiendo de unos valores, fácilmente modificables, que cambiaremos en cada aplicación en la que deseemos utilizar una rutina que hayamos realizado previamente.

Un programa se llama *macro*-ensamblador y no simplemente ensamblador, cuando dispone de la utilidad de macros.

Esta permite que el programa ensamblador se encargue de repetir zonas de programa o datos, librando de este pesado trabajo al programador. A continuación ofrecemos un breve resumen de las directivas disponibles en el macro-ensamblador MACRO-80.



PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS

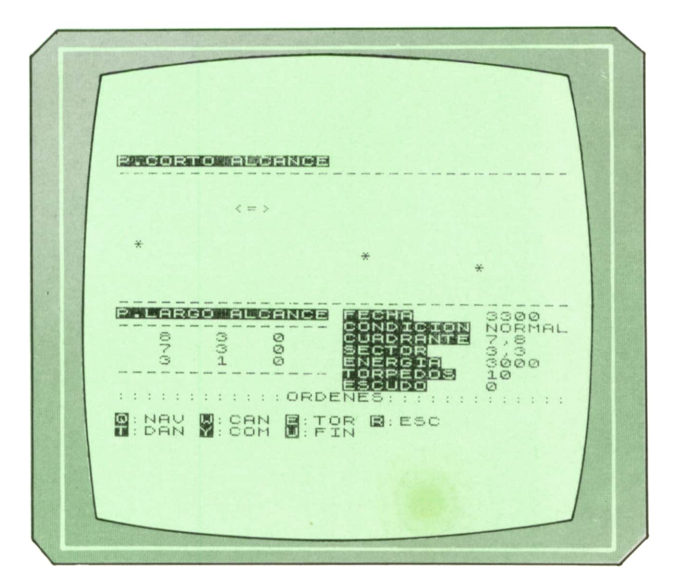




Fig. 1. Pantalla de juego y comandos.

Programa: STAR TREK

EL juego que vamos a ver a continuación es una versión del famoso STAR TREK. Este fue uno de los primeros juegos que apareció para ordenador y, aunque ha pasado mucho tiempo, se puede decir que es uno de los mejores, ya que en él se mezcla la acción con la inteligencia.

```
  1 REM *************
  2 REM * STAR TREK *
  3 REM *************
  4 REM
  5 REM ********************
  6 REM * POR:             *
  7 REM *                  *
  8 REM * MANUEL ALFONSECA *
  9 REM * Y                *
 10 REM * Fco. MORALES     *
 11 REM ********************
 12 REM
 13 REM ********************************
 14 REM * (c) Ediciones Siglo Cultural *
 15 REM * (c) 1987                     *
 16 REM ********************************
 17 REM
 20 LET x=0: LET xh=0: LET XK=0: CLS : PRINT "       ESPERA UN MOMENTO."''''''"
ESTOY CREANDO LA GALAXIA."
 30 DIM G(8,8)
 40 DIM Q$(8,32): REM MAPA DEL CUADRANTE DE LA GALAXIA"
 50 DIM Z(10,10): REM MAPA ACUMULADO DE LA GALAXIA"
 60 DIM F(8):       REM DANOS EN LA NAVE AVENTURA
 70 DIM K(10,3):    REM NAVES ENEMIGAS EN ESTE CUADRANTE
 80 DIM I(1,2):     REM BASES ESTELARES EN ESTE CUADRANTE
 90 DIM E(8,2):     REM ESTRELLAS EN ESTE CUADRANTE
```

```
  91 REM
  95  REM VARIABLES C UTILIZADAS POR TORPEDO
  99 REM
 100 DIM A(8): DIM B(8): DIM C(8): DIM D(8)
 110 LET A(1)=0: LET A(2)=-1: LET A(3)=1: LET A(4)=-1: LET A(5)=0: LET A(6)=0: L
ET A(7)=1: LET A(8)=1
 120 LET B(1)=-1: LET B(2)=0: LET B(3)=0: LET B(4)=1: LET B(5)=1: LET B(6)=0: LE
T B(7)=0: LET B(8)=-1
 130 LET C(1)=1: LET C(2)=1: LET C(3)=0: LET C(4)=-1: LET C(5)=-1: LET C(6)=-1:
LET C(7)=0: LET C(8)=1
 140 LET D(1)=0: LET D(2)=-1: LET D(3)=-1: LET D(4)=0: LET D(5)=0: LET D(6)=1: L
ET D(7)=1: LET D(8)=0
1000 RANDOMIZE
1001 REM
1002 REM ***************************************
1003 REM * COMIENZA LA CREACION DE LA AVENTURA *
1004 REM ***************************************
1005 REM
1006 REM ***************************************************
1007 REM * GENERACION DE ESTRELLAS, BASES Y NAVES ENEMIGAS *
1008 REM ***************************************************
1009 REM
1010 LET B9=0: LET K9=0
1020 FOR I=1 TO 8: FOR J=1 TO 8: LET A=RND
1030 LET AK=0
1040 IF A>.8 THEN  LET AK=1
1050 IF A>.95  THEN  LET AK=2
1060 IF A>9799999 THEN  LET AK=3
1070 LET K9=K9+AK
1080 LET A=RND
1090 LET AB=0
1100 IF A>.96 THEN  LET AB=1
1110 LET B9=B9+AB
1120 LET AS=1+INT (8*RND)
1130 LET G(I,J)=AS+10*AB+100*AK
1140 NEXT J: NEXT I
1150 LET K7=K9
1191 REM
1192 REM ********************************
1200  REM * POSICION DE LA NAVE AVENTURA *
1201 REM ********************************
1202 REM
1210 LET Q1=1+INT (8*RND): LET Q2=1+INT (8*RND)
1220 LET S1=1+INT (8*RND): LET S2=1+INT (8*RND)
1221 REM
1222 REM **********
1230 REM * FECHAS *
1232 REM **********
1233 REM
1240 LET T0=100*(20+INT (20*RND)): REM FECHA INICIAL
1250 LET T=T0: REM FECHA ACTUAL
1260 LET T9=30: IF K9>20 THEN  LET T9=30: REM FECHA FINAL
1261 REM
1262 REM ***************
1270 REM * OTROS DATOS *
1272 REM ***************
1273 REM
1280 LET E0=3000: REM ENERGIA INICIAL
1290 LET E1=E0:     REM ENERGIA ACTUAL
1300 LET P0=10:     REM NUMERO INICIAL DE TORPEDOS
1310 LET P1=P0:     REM NUMERO ACTUAL DE TORPEDOS
1320 LET S0=0:      REM ESCUDO PROTECTOR ACTUAL
1321 REM
1322 REM ************************
1500 REM * COMIENZA LA AVENTURA *
1502 REM ************************
1503 REM
```

```
1510 CLS : PRINT "AL CAPITAN DE LA NAVE AVENTURA. ESTAS SON LAS ORDENES:"
1515 PRINT
1520 PRINT "DEBE USTED DESTRUIR LAS ";K9;" NAVESENEMIGAS QUE HAN INVADIDO LA GA-
"
1530 PRINT "LAXIA, ANTES DE  QUE PUEDAN ATA-"
1540 PRINT "CAR EL CUARTEL GENERAL DE LA FE-DERACION EN LA FECHA ESTELAR"
1550 PRINT TO-T9;". ESTO LE DEJA ";T9;" FECHAS."
1555 PRINT
1560 PRINT "HAY ";B9;" BASES ESTELARES  EN LA GA- LAXIA, DONDE PUEDE  REABASTACE
R"
1570  PRINT "LA NAVE AVENTURA. BUENA SUERTE!"
1580 PRINT : INPUT "PRESIONE ENTER CUANDO ESTE DISPUESTO A ASUMIR EL MANDO.";A$
1600 CLS : GO SUB 1610: GO TO 2000: REM INICIALIZAR PANTALLA
1610 PRINT AT 0,0; INVERSE 1;"P.CORTO ALCANCE"
1650 PRINT AT 13,16; INVERSE 1;"FECHA"
1680 PRINT AT 14,16; INVERSE 1;"CONDICION"
1700 PRINT AT 15,16; INVERSE 1;"CUADRANTE"
1720 PRINT AT 16,16; INVERSE 1;"SECTOR"
1740 PRINT AT 17,16; INVERSE 1;"ENERGIA"
1760 PRINT AT 18,16; INVERSE 1;"TORPEDOS"
1780 PRINT AT 19,16; INVERSE 1;"ESCUDO"
1790  RETURN
1791 REM
1792 REM *******************
2000 REM * SALA DE CONTROL *
2001 REM *******************
2002 REM
2010 GO SUB 2020: GO TO 2200
2020 GO SUB 1610: PRINT AT 13,0; INVERSE 1;"P.LARGO ALCANCE"
2030 IF F(3)<0  THEN  GO TO 2150
2040 PRINT "---------------"
2050 FOR I=1 TO 3: PRINT AT 14+I,0;: FOR J=1  TO 3
2060 LET A=0
2065 LET A1=Q1+I: LET A2=Q2+J
2070 IF (A1>2) AND (A1<11) AND (A2>2) AND (A2<11) THEN  LET A=G(A1-2,A2-2)
2080 LET Z(Q1+I-1,Q2+J-1)=A: PRINT ("    "+STR$ (A))(1+(A>9)+(A>99)  TO 4+(A>9)+(
A>99));
2090 NEXT J: NEXT I
2100 PRINT AT 18,0;"---------------":  RETURN
2150 GO SUB 9947: PRINT : PRINT  FLASH 1;"P. LARGO ALCANCE ESTROPEADA"
2160 FOR I=1 TO 5: PRINT AT 13+I,0;: FOR J=0  TO 15
2170 PRINT " ";
2180 NEXT J: NEXT I: RETURN
2181 REM
2182 REM **********************************
2200 REM * ENTRAMOS EN UN NUEVO CUADRANTE *
2201 REM **********************************
2202 REM
2210 PRINT AT 15,26;Q1;",";Q2
2230 LET S3=G(Q1,Q2): LET K3=INT (S3/100): LET S3=S3-100*K3: LET B3=INT (S3/10):
 LET S3=S3-10*B3
2231 REM
2232 REM ********************************
2240 REM * DISTRIBUCION EN EL CUADRANTE *
2241 REM ********************************
2242 REM
2250 FOR I=1 TO 8: FOR J=1 TO 32: LET Q$(I,J)=" ": NEXT J: NEXT I
2260 LET Q$(S1,1+4*(S2-1))="<"
2270 LET Q$(S1,2+4*(S2-1))="="
2280 LET Q$(S1,3+4*(S2-1))=">"
2290 LET V$=".": LET W$=".": LET X$="."
2300 FOR I=1 TO K3: GO SUB 2400: LET K(I,1)=R1: LET K(I,2)=R2: LET K(I,3)=200: N
EXT I
2303 IF K3=3 THEN  GO TO 2310
2305 FOR I=K3+1 TO 3: LET K(I,3)=0: NEXT I
2310 LET V$="-": LET W$="O": LET X$="-"
```

```
2320 FOR I=1  TO B3: GO SUB 2400: LET I(I,1)=R1: LET I(I,2)=R2: NEXT I
2330 LET V$=" ": LET W$="*": LET X$=" "
2340 FOR I=1 TO S3: GO SUB 2400: LET E(I,1)=R1: LET E(I,2)=R2: NEXT I
2350 GO TO 2500
2400 LET R1=1+INT (8*RND): LET R2=1+INT (8*RND)
2410 IF Q$(R1,2+4*(R2-1))<>" " THEN  GO TO 2400
2420 LET Q$(R1,1+4*(R2-1))=V$
2430 LET Q$(R1,2+4*(R2-1))=W$
2440 LET Q$(R1,3+4*(R2-1))=X$
2450 RETURN
2451 REM
2452 REM ****************************************************
2500 REM * COMPROBACION DE POSICION DE AMARRE CON ESTACION *
2501 REM ****************************************************
2502 REM
2510 IF B3=0 THEN  GO TO 2600: REM NO HAY BASE
2520 IF S2<>I(1,2) THEN  GO TO 2600: REM NO ESTAMOS AMARRADOS
2530 IF (S1<I(1,1)-1) OR (S1>I(1,1)+1) THEN  GO TO 2600
2540 PRINT AT 14,26; FLASH 1;"PUERTO"
2550 LET D0=1: REM EN PUERTO
2560 LET E1=E0: LET P1=P0: LET S0=0: REM REPOSTAMOS
2570 GO SUB 9947: PRINT "SIN  ESCUDO  PROTECTOR  PARA  ELAMARRE."
2580 GO TO 2750
2600 LET D0=0: REM NO ESTAMOS AMARRADOS
2610 IF K3>0 THEN  GO TO 2650
2620 IF E1<.1*E0 THEN  GO TO 2700
2630 PRINT AT 14,26;"NORMAL"
2640 GO TO 2750
2650 PRINT AT 14,26; FLASH 1;"ROJA "
2660 BEEP 1,15
2670 IF S0>200 THEN  GO TO 2750
2680 GO SUB 9947: PRINT : PRINT  FLASH 1;"   ESCUDO PROTECTOR MUY BAJO.   "
2690 GO TO 2750
2700 PRINT AT 14,26; INVERSE 1;"AMBAR "
2750 IF F(2)<0 THEN  GO TO 2800: REM PANTALLA DE CORTO ALCANCE
2760 PRINT AT 1,0;"--------------------------------"
2770 FOR I=1 TO 8: PRINT AT 1+I,0;
2780 FOR J=1 TO 32: PRINT Q$(I,J);: NEXT J: NEXT I
2790 GO TO 2820
2800 GO SUB 9947: PRINT  FLASH 1;"    P.CORTO ALCANCE ESTROPEADA    "
2810 FOR I=1 TO 8: PRINT AT 1+I,0;: FOR J=1 TO 32: PRINT " ";: NEXT J: NEXT I
2820 PRINT AT 12,0;"--------------------------------"
2830 PRINT AT 13,26;"      ";AT 13,26;T
2840 PRINT AT 16,26;"      ";AT 16,26;S1;",";S2
2850 PRINT AT 17,26;"      ";AT 17,26;E1+S0
2860 PRINT AT 18,26;"      ";AT 18,26;P1
2870 PRINT AT 19,26;"      ";AT 19,26;S0
2871 REM
2872 REM *****************************
2900 REM * COMPROBACION DE COMBUSTIBLE *
2901 REM *******************************
2902 REM
2910 IF S0+E1<10 THEN  GO TO 2930
2920 IF (E1>10) OR (F(7)=0) THEN  GO TO 3000
2930 BEEP 1,15
2940 CLS : PRINT "ERROR FATAL!!!"'''"LA NAVE HA QUEDADO VARADA EN ELESPACIO."
2950 PRINT '"NO HAY ENERGIA SUFICIENTE, Y EL CONTROL DE ESCUDO NO PUEDE PASAR"
2960 PRINT "ENERGIA A LA SALA DE MAQUINAS."
2970 GO TO 9994
2971 REM
2972 REM ********************************
2990 REM * BUCLE DE PETICION DE ORDENES *
2991 REM ********************************
2992 REM
3000 GO SUB 9947: PRINT AT 20,0;"::::::::::::ORDENES:::::::::::::
     "
3030 PRINT #0;AT 0,0; INVERSE 1;"Q"; INVERSE 0;":NAV "; INVERSE 1;"W"; INVERSE 0
```

```
;":CAN "; INVERSE 1;"E"; INVERSE 0;":TOR "; INVERSE 1;"R"; INVERSE 0;":ESC
   "; INVERSE 1;"T"; INVERSE 0;":DAN "; INVERSE 1;"Y"; INVERSE 0;":COM "; INVERS
E 1;"U"; INVERSE 0;":FIN";
3040 GO SUB 9997: REM LEE UNA TECLA
3060 IF I=CODE "U" THEN  CLS : GO TO 9994: REM FIN
3080 IF I=CODE "Q" THEN  GO TO 4000
3090 IF I=CODE "W" THEN  GO TO 5000
3100 IF I=CODE "E" THEN  GO TO 6000
3110 IF I=CODE "R" THEN  GO TO 7000
3120 IF I=CODE "T" THEN  GO TO 8000
3130 IF I=CODE "Y" THEN  GO TO 9000
3140 GO TO 3040
3141 REM
3142 REM **************************
4000 REM * CONTROL DE NAVEGACION *
4001 REM **************************
4010 INPUT "CURSO (1-9): ";C1: IF C1<1 OR C1>9 THEN  GO TO 4010
4030 INPUT "VELOCIDAD (0-8): ";W1: IF W1<0 OR W1>8 THEN  GO TO 4030
4050 IF C1=9 THEN  LET C1=1
4060 IF (W1<=.2) OR (F(1)>=0) THEN  GO TO 4100
4070 GO SUB 9947: PRINT "MOTORES DANADOS.";#0;AT 0,0;"VELOCIDAD MAXIMA=0.2":  GO
 SUB 9950
4090 GO TO 4000
4100 LET N=INT (.5+8*W1): IF E1>=N  THEN  GO TO 4150
4110 GO SUB 9947: PRINT  FLASH 1;"ENERGIA INSUFICIENTE PARA        MANIOBRAR": GO
 SUB 9950
4120 IF (F(7)<0) OR (S0<N-E1) THEN  GO TO 3000
4130  GO SUB 9947:  PRINT S0;"UNIDADES DISPONIBLES PARA ESCUDO":  GO SUB 9950
4140 GO TO 3000
4141 REM
4142 REM ***************************
4150 REM * COMIENZA EL MOVIMIENTO *
4151 REM ***************************
4152 REM
4153 REM *****************************
4160 REM * PRIMERO ATACA EL ENEMIGO *
4161 REM *****************************
4162 REM
4170 GO SUB 9915
4171 REM
4172 REM ***************************
4180 REM * REPARACIONES EN MARCHA *
4181 REM ***************************
4182 REM
4190 GO SUB 9932
4200 LET Z1=S1: LET Z2=S2
4210 LET Q$(INT (S1),1+4*(INT (S2)-1))=" "
4220 LET Q$(INT (S1),2+4*(INT (S2)-1))=" "
4230 LET Q$(INT (S1),3+4*(INT (S2)-1))=" "
4240 LET X1=A(INT (C1))+B(INT (C1))*(C1-INT (C1))
4250 LET X2=C(INT (C1))+D(INT (C1))*(C1-INT (C1))
4260 LET I=1
4270 LET S1=S1+X1: LET S2=S2+X2
4280 IF (S1<1) OR (S1>=9) OR (S2<1) OR (S2>=9) THEN  GO TO 4400
4290 IF Q$(INT (S1),2+4*(INT (S2)-1))=" " THEN  GO TO 4340
4300 LET S1=S1-X1: LET S2=S2-X2
4310 GO SUB 9947: PRINT  FLASH 1;"PARADA AUTOMATICA DE MOTORES    DEBIDO A MALA
NAVEGACION        ";
4330 GO TO 4350
4340 LET I=I+1: IF N>=I THEN  GO TO 4270
4350 LET Q$(INT (S1),1+4*(INT (S2)-1))="<"
4360 LET Q$(INT (S1),2+4*(INT (S2)-1))="="
4370 LET Q$(INT (S1),3+4*(INT (S2)-1))=">"
4380 GO SUB 4900
4390 LET S1=INT (.5+S1):  LET S2=INT (.5+S2):  GO TO 2500
4400 LET Z1=(8*Q1)+Z1+N*X1
4410 LET Z2=(8*Q2)+Z2+N*X2
4420 LET Q1=INT (Z1/8): LET S1=INT (Z1-8*Q1)
```

```
4430 LET Q2=INT (Z2/8): LET S2=INT (Z2-8*Q2)
4440 IF S1=0 THEN  LET S1=8: LET Q1=Q1-1
4450 IF S2=0 THEN  LET S2=8: LET Q2=Q2-1
4460 IF Q1<1 THEN  LET Q1=1: GO SUB 4800
4470 IF Q2<1 THEN  LET Q2=1: GO SUB 4800
4480 IF Q1>8 THEN  LET Q1=8: GO SUB 4800
4490 IF Q2>8 THEN  LET Q2=8: GO SUB 4800
4500 GO SUB 4900
4510 LET S1=INT (.5+S1): LET S2=INT (.5+S2): GO TO 2000
4800 GO SUB 9947: PRINT  FLASH 1;"DETENCION AUTOMATICA DE MOTORES";#0;AT 0,0; FL
ASH 0;"PERMISO DENEGADO PARA ABANDONAR LA GALAXIA"
4830 RETURN
4900 LET E1=E1-N-10
4910 IF E1>=0 THEN  GO TO 4960
4920 GO SUB 9947: PRINT "ENERGIA TRANSFERIDA DESDE EL ES-CUDO PARA COMPLETAR LA
MANIOBRA"
4940 LET S0=S0+E1: IF S0<0  THEN  LET S0=0
4950 LET E1=0
4960 LET TT1=.1*INT (10*W1): IF                 LET TT1=1: LET T=T+TT1
4970 IF T>T0+T9 THEN  GO TO 9994
4980 RETURN
4981 REM
4982 REM ****************************
5000 REM * DISPARO DE CANONES LASER *
5001 REM ****************************
5002 REM
5010 IF K3>0 THEN  GO TO 5040
5020 GO SUB 9947: PRINT : PRINT "NO HAY NAVES ENEMIGAS A LA VISTA":  GO SUB 9950
5030 GO TO 3000
5040 IF F(4)>=0  THEN  GO TO 5070
5050 GO SUB 9947: PRINT  FLASH 1;"EL CANON LASER NO FUNCIONA"
5060  GO TO 3000
5070 GO SUB 9947: PRINT "CANON LASER APUNTANDO A OBJETIVO"
5090 PRINT  INVERSE 1;"ENERGIA DISPONIBLE=";E1
5100 IF F(8)>=0 THEN  GO TO 5120
5110 GO TO 9947: PRINT "MAYOR PRECISION POR FALLO COMPUTADOR"
5120 INPUT "ENERGIA DISPARO:";C1: IF C1<0 OR C1>E1 THEN  GO TO 5120
5140 LET E1=E1-C1
5150 GO SUB 9915
5160 IF F(7)>=0 THEN  GO TO 5180
5170 LET C1=C1*RND
5180 LET C1=INT (C1/K3)
5190 FOR I=1  TO 3
5200 IF K(I,3)=0 THEN  GO TO 5260
5210 LET H=INT ((2*RND)*C1/SQR (((K(I,1)-S1)*(K(I,1)-S1))+((K(I,2)-S2)*(K(I,2)-S
2))))
5220 IF H>=.15*K(I,3) THEN  GO TO 5260
5230 LET K(I,3)=K(I,3)-H
5240 GO SUB 9947:  PRINT H;" UNIDADES PARA ENEMIGO EN ";K(I,1);",";K(I,2)
5250 IF K(I,3)<0 THEN  LET XA=K(I,1): LET XB=K(I,2): GO SUB 9890
5251 REM
5252 REM *****************************
6000 REM * LANZAMIENTO DE UN TORPEDO *
6001 REM *****************************
6002 REM
6010 GO SUB 9947
6020 IF F(5)<0 THEN  PRINT "LOS TUBOS DE TORPEDOS NO FUNCIONAN":  GO TO 3000
6030 IF P1<=0 THEN  PRINT "TORPEDOS AGOTADOS":  GO TO 3000
6040 INPUT "CURSO TORPEDO (1-9) ";N1: IF N1<1 OR N1>9 THEN  GO TO 6040
6060 IF N1=9 THEN  LET N1=1
6070 LET X1=A(INT (N1))+B(INT (N1))*(N1-INT (N1))
6080 LET X2=C(INT (N1))+D(INT (N1))*(N1-INT (N1))
6090 LET E1=E1-2:  LET P1=P1-1:  LET XA=S1:  LET XB=S2
6100 PRINT AT 21,0;"CURSO DE TORPEDO: ";
6110 LET XA=XA+X1: LET XB=XB+X2
6120 IF (XA>=9) OR (XB>=9) OR (XA<1) OR  (XB<1) THEN  GO TO 6300
6130 PRINT AT 21,19;XA;",";XB
```

```
6140 LET I$=Q$(INT (XA+.5),2+4*INT (XB-.5))
6150 IF I$=" " THEN  GO TO 6310
6160 IF I$="." THEN  GO SUB 9890:  GO TO 6250
6170 IF I$="*" THEN  GO TO 6290
6180 GO SUB 9947: PRINT  INVERSE 1;"BASE ESTELAR DESTRUIDA, ESTUPIDO"
6190 LET B3=B3-1: LET B9=B9-1: LET D0=0
6200 LET Q$(INT (XA),1+4*INT (XB-1))=" "
6210 LET Q$(INT (XA),2+4*INT (XB-1))=" "
6220 LET Q$(INT (XA),3+4*INT (XB-1))=" "
6230 LET G(Q1,Q2)=S3+10*(B3+10*K3)
6240 GO SUB 2020
6250 GO SUB 9915
6280 GO TO 2500
6290 GO SUB 9947: PRINT "UNA ESTRELLA ABSORBIO EL TORPEDO":  GO TO 6250
6300 GO SUB 9947: PRINT  INVERSE 1;"EL TORPEDO HA FALLADO EL OBJETIVO":  GO TO 6
250
6310 IF XH<>0 THEN  PRINT AT 3+XH,XK:  PRINT " "
6320 LET XH=INT (XA+.5): LET XK=2+4*INT (XB-.5)
6330 PRINT AT 3+XH,XK;"": BEEP 1,15: GO TO 6110
6331 REM
6332 REM ******************************************
7000 REM * TRASPASO DE ENERGIA AL ESCUDO PROTECTOR *
7001 REM ******************************************
7002 REM
7010 GO SUB 9947
7020 IF F(7)>=0 THEN  GO TO 7030
7025 GO SUB 9947: PRINT : PRINT  FLASH 1;"EL ESCUDO PROTECTOR NO FUNCIONA":  GO
TO 3000
7030 INPUT "ENERGIA AL ESCUDO";N1: IF N1<0 OR N1>E1+S0 THEN  GO TO 7030
7050 LET E1=E1+S0-N1
7060 LET S0=N1
7070 GO SUB 9947: PRINT : PRINT "ESCUDO A ";S0;" SEGUN SUS ORDENES"
7080 PRINT AT 19,26;"      ";AT 19,26;S0: GO SUB 9950: GO TO 3000
7081 REM
7082 REM *******************
8000 REM * INFORME DE DANOS *
8001 REM *******************
8002 REM
8005 FOR I=1 TO 10: PRINT AT I,0;"                                ": NEXT I
8010 GO SUB 9947
8020 IF F(6)<0 THEN  PRINT "EL CONTROL DE DANOS NO FUNCIONA":  GO TO 8110
8030 PRINT AT 0,0; INVERSE 1;"CONTROL DE DANOS":  FOR I=1  TO 8
8040 PRINT AT I,0;: GO SUB 8900: PRINT D$;F(I)
8050 NEXT I: GO SUB 9950
8110 IF D0=0 THEN  GO TO 8300
8120 LET D3=0: FOR I=1  TO 8: IF F(I)<0  THEN  LET D3=D3+.1
8130 NEXT I: IF D3>1 THEN  LET D3=1
8140 IF D3=0 THEN  GO TO 8300
8150 GO SUB 9947: PRINT "TECNICOS DISPUESTOS A REPARAR NAVE"
8160 PRINT #0;AT 0,0;"TIEMPO DE REPARACION ESTIMADO:";D3;"FECHAS ESTELARES":  GO
 SUB 9950
8180 GO SUB 9947: INPUT "AUTORIZA USTED LAS REPARACIONES S/N";I$:  IF I$<>"SI"
AND I$<>"NO"  THEN  GO TO 8180
8190 IF I$="NO" THEN  GO TO 2500
8210 FOR I=1 TO 8: LET F(I)=0: NEXT I
8230 LET T=T+D3+.1: GO TO 8030
8320 FOR I=0  TO 10: PRINT AT I,0;"                                ":  NEXT I:
GO SUB 2020:  GO TO 2500
8900 GO TO 8900+10*I
8910 LET D$="MOTORES............":  RETURN
8920 LET D$="PANTALLA CORTO AL...":  RETURN
8930 LET D$="PANTALLA LARGO AL...":  RETURN
8940 LET D$="CANONES LASER.......":  RETURN
8950 LET D$="TUBOS DE TORPEDOS...":  RETURN
8960 LET D$="CONTROL DE DANOS....": RETURN
8970 LET D$="CONTROL DE ESCUDO...": RETURN
8980 LET D$="COMPUTADORA.........": RETURN
```

```
9000 REM COMPUTADORA DE A BORDO
9010 GO SUB 9947
9020 IF F(8)<0 THEN  GO TO 9150
9030 PRINT ":::::::COMPUTADOR ACTIVO.:::::::                    "
9050 PRINT #0;AT 0,0; INVERSE 1;"Q"; INVERSE 0;":GALAXIA "; INVERSE 1;"W"; INVER
SE 0;":SITUACION "; INVERSE 1;"E"; INVERSE 0;":TORPEDOS"; INVERSE 1;"R"; INVERSE
 0;":BASE     "; INVERSE 1;"T"; INVERSE 0;":DIR-DIST";
9060 GO SUB 9997: REM LEE UNA TECLA
9070 IF I=CODE "Q" THEN  GO TO 9200
9080 IF I=CODE "W" THEN  GO TO 9400
9090 IF I=CODE "E" THEN  GO TO 9500
9100 IF I=CODE "R" THEN  GO TO 9700
9110 IF I=CODE "T" THEN  GO TO 9800
9120 GO TO 9060
9150 PRINT  FLASH 1;"LA COMPUTADORA NO FUNCIONA":  GO TO 3000
9151 REM
9152 REM **********************
9200 REM * MAPA DE LA GALAXIA *
9201 REM **********************
9202 REM
9205 FOR I=1 TO 9: PRINT AT I,0;"                                  ": NEXT I
9210 PRINT AT 0,0; INVERSE 1;"  MAPA ACUMULADO DE LA GALAXIA  "
9220 PRINT AT 1,0;"   "; INVERSE 1;"1"; INVERSE 0;"   "; INVERSE 1;"2"; INVERSE 0
;"   "; INVERSE 1;"3"; INVERSE 0;"   "; INVERSE 1;"4"; INVERSE 0;"   "; INVERSE 1;"
5"; INVERSE 0;"   "; INVERSE 1;"6"; INVERSE 0;"   "; INVERSE 1;"7"; INVERSE 0;"   "
; INVERSE 1;"8"
9230 PRINT AT 2,0;"----------------------------"
9240 FOR I=1  TO 8: PRINT AT 2+I,0; INVERSE 1;I;
9250 FOR J=1 TO 8: PRINT ("    "+STR$ (Z(I+1,J+1)))(2+(Z(I+1,J+1)>9)+(Z(I+1,J+1)>
99)  TO 4+(Z(I+1,J+1)>9)+(Z(I+1,J+1)>99));:  NEXT J:  NEXT I
9260 PRINT AT 12,0;"PRESIONE UNA TECLA PARA SEGUIR"
9270 LET A$=INKEY$: IF A$=""  THEN  GO TO 9270
9330 CLS : GO SUB 2020: GO TO 2500
9331 REM
9332 REM *************
9400 REM * SITUACION *
9401 REM *************
9402 REM
9410 PRINT AT 0,0;: FOR I=1  TO 10:  PRINT "                                ";AT
 0,0; INVERSE 1;"      INFORME DE SITUACION      "
9420 PRINT : PRINT ;"QUEDAN ";K9;" NAVES ENEMIGAS"
9430 PRINT "QUEDAN ";T0+T9-T;" FECHAS ESTELARES"
9440 PRINT "HAY ";B9;" BASES ESTELARES EN LA GALAXIA"
9445 PRINT AT 10,0; INVERSE 1;" PULSE UNA TECLA PARA CONTINUAR ": IF INKEY$=""
THEN  GO TO 9445
9450 CLS : GO SUB 2020: GO SUB 2500
9451 REM
9452 REM **************************************
9500 REM * CALCULO DE DIRECCION PARA TORPEDOS *
9501 REM **************************************
9502 REM
9505 GO SUB 9947
9510 IF K3=0 THEN  PRINT : PRINT "NO HAY NAVES ENEMIGAS AQUI":  GO SUB 9950:  GO
 TO 3000
9512 FOR I=1  TO 3
9515 IF K(I,3)=0 THEN  GO TO 9525
9520 LET N1=K(I,1): LET N2=K(I,2): GO SUB 9530
9525 NEXT I: GO TO 3000
9526 REM
9527 REM ********************************************
9528 REM * RUTINA DE CALCULO DE DIRECCION/DISTANCIA *
9529 REM ********************************************
9530 REM
9535 PRINT "DIRECCION        DISTANCIA"
9540 LET X1=N1: LET X2=N2
9550 LET A=S1-X1: LET X=X2-S2
9560 IF X<0 THEN  GO TO 9610
```

```
9562 IF A<0 THEN  GO TO 9640
9570 IF X>0 THEN  GO TO 9580
9572 IF A=0 THEN  GO TO 9620
9580 LET C1=1
9590 IF ABS (A)>ABS (X)  THEN  LET C1=C1+2-ABS (X/A):  GO TO 9670
9595 LET C1=C1+ABS (A/X)
9600 GO TO 9670
9610 IF A>0  THEN  GO TO 9630
9615 IF X=0  THEN  GO TO 9640
9620 LET C1=5:  IF (A=0)  AND (X=0)  THEN  GO TO 9670
9625 GO TO 9590
9630 LET C1=3:  GO TO 9650
9640 LET C1=7
9650 IF ABS (A)<ABS (X)  THEN  LET C1=C1+2-ABS (A/X):  GO TO 9670
9660 LET C1=C1+ABS (X/A)
9670 PRINT C1,SQR (X*X+A*A):  GO SUB 9950:  RETURN
9671 REM
9672 REM ***********************************
9700 REM * CALCULO DE DIRECCION PARA BASES *
9701 REM ***********************************
9702 REM
9710 GO SUB 9947
9720 IF B3=0  THEN  GO TO 9790
9730 LET N1=I(1,1): LET N2=I(1,2):  GO SUB 9530:  GO TO 3000
9790 PRINT "NO HAY BASES ESTELARES AQUI":  GO SUB 9950:  GO TO 3000
9791 REM
9792 REM ****************************************
9800 REM * CALCULO DE UNA DIRECCION CUALQUIERA *
9801 REM ***************************
9802 REM
9810 INPUT "COORDENADA X ";X1:  IF X1<1  OR X1>8  THEN  GO TO 9810
9820 INPUT "COORDENADA Y ";X2:  IF X2<1  OR X2>8  THEN  GO TO 9820
9830 LET N1=X1:  LET N2=X2:  LET A=Q1-X1:  LET X=X2-Q2
9870 GO SUB 9947:  PRINT "DIRECCION        DISTANCIA"
9880 LET I=1:  GO SUB 9560:  GO SUB 9950:  GO TO 3000
9881 REM
9882 REM ***************************
9890 REM * NAVE ENEMIGA DESTRUIDA *
9891 REM ***************************
9892 REM
9893 FOR I=-10 TO 0:  BEEP .1,I:  NEXT I
9894 GO SUB 9947: PRINT : PRINT "NAVE ENEMIGA DESTRUIDA"
9896 LET K3=K3-1:  LET K9=K9-1:  IF K9=0  THEN  GO TO 9910
9898 FOR L=1  TO 3:  IF KA<>K(L,1)  THEN  GO TO 9900
9899 IF KB=K(L,2)  THEN  GO TO 9901
9900 NEXT L
9901 LET K(L,3)=0
9902 LET Q$(INT (XA+.5),1+4*INT (XB-.5))=" "
9903 LET Q$(INT (XA+.5),2+4*INT (XB-.5))=" "
9904 LET Q$(INT (XA+.5),3+4*INT (XB-.5))=" ":  LET G(Q1,Q2)=S3+10*(B3+10*K3)
9905 GO SUB 2020:  RETURN
9910 CLS :  PRINT "ENHORABUENA, CAPITAN! LA ULTIMA NAVE ENEMIGA"
9911 PRINT "HA SIDO DESTRUIDA":  PRINT
9912 PRINT "SU EFICIENCIA ES IGUAL A";INT (1000*K7/(T-T0)):  STOP
9915 REM *** ATAQUE ENEMIGO ***
9916 IF K3=0  THEN  RETURN
9917 IF D0=0  THEN  GO TO 9919
9918 GO SUB 9947:  PRINT "LA BASE PROTEGE LA NAVE AVENTURA":  RETURN
9919 FOR L=1  TO 3:  IF K(L,3)=0  THEN  GO TO 9931
9920 LET H=SQR (((K(L,1)-S1)*(K(L,1)-S1))+((K(L,2)-S2)*(K(L,2)-S2)))
9921 LET H=INT ((2+RND)*K(L,3)/H)
9922 IF H=0  THEN  GO TO 9931
9923 LET S0=S0-H:  GO SUB 9947:  PRINT H;" UNIDADES ALCANZAN AL AVENTURA DESDE E
L SECTOR ";K(L,1);",";K(L,2)
9925 IF S0<0  THEN  CLS :  PRINT "EL AVENTURA HA SIDO DESTRUIDO":  GO TO 9994
9926 PRINT #0;AT 0,0;" ESCUDO DISMINUYE A ";S0
9927 IF (H<20)  OR (.02>=H/S0)  OR (RND>.5)  THEN  GO TO 9931
```

```
9928 LET I=INT (1+RND*8):  LET F(I)=F(I)-(H/SO)-.5*RND
9929 BEEP .5,-10:  BEEP .5,-15:  GO SUB 9947:  PRINT "CONTROL DE DANOS INFORMA:"
9930 GO SUB 8900:  PRINT #0;AT 0,0;D$;"DANADO"
9931 GO SUB 9950:  NEXT L:  RETURN
9932 REM *** REPARACIONES EN MARCHA ***
9933 LET D1=14:  LET D6=W1:  IF D6>1  THEN  LET D6=1
9934 FOR I=1  TO 8:  IF F(I)=0  THEN  GO TO 9939
9935 LET F(I)=F(I)+D6:  IF F(I)<0  THEN  GO TO 9939
9936 IF F(I)>0  THEN  LET F(I)=0
9937 GO SUB 9947:  PRINT #0;AT 0,0;: GO SUB 8900
9938 PRINT D$;"REPARADO":  GO SUB 9950
9939 NEXT I
9940 IF RND>.1  THEN  RETURN
9941 LET I=1+INT (RND*8)
9942 LET F(I)=F(I)-.1-RND*5
9943 GO SUB 8900
9944 GO SUB 9947:  PRINT "CONTROL DE DATOS INFORMA:"
9945 PRINT #0;AT 0,0;D$;"DANADO"
9946 RETURN
9947 REM *** BORRA ZONA DE INFORMACION ***
9948 PRINT AT 20,0;"                                                  ";#0;AT 0,0;"
                               ";
9949 PRINT AT 20,0;: RETURN
9950 REM *** RETARDO ***
9951 FOR I=1  TO 40:  LET J=2.2*I:  NEXT I:  RETURN
9994 PRINT "ES LA FECHA ESTELAR ";T
9995 PRINT "QUEDAN ";K9;" NAVES ENEMIGAS EN LA GALAXIA AL FINAL DE SU MISION."
9996 PRINT "LA FEDERACION SERA CONQUISTADA.":  STOP
9997 LET A$=INKEY$:  IF A$=""  THEN  GO TO 9997
9998 LET I=CODE (A$):  RETURN
```

Este programa está pensado para que funcione en el SPECTRUM, pero más adelante aparecerá la versión para IBM y AMSTRAD.

El objetivo del juego es destruir a todos los invasores de la galaxia que planean hacerse con el control de la federación.

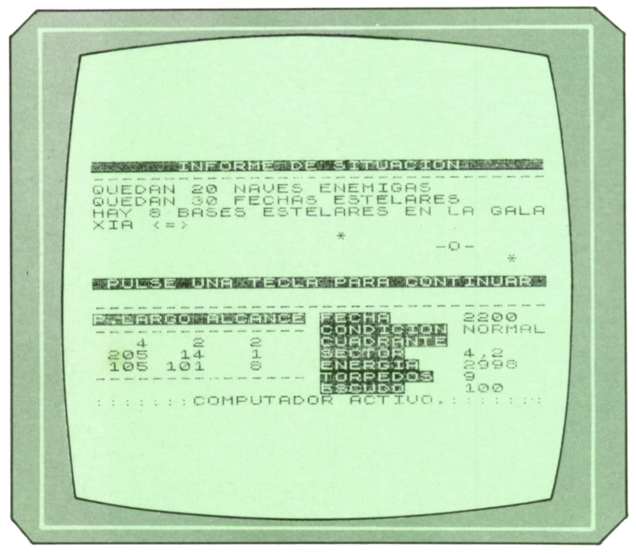



Fig. 2. Informe de situación en la galaxia.

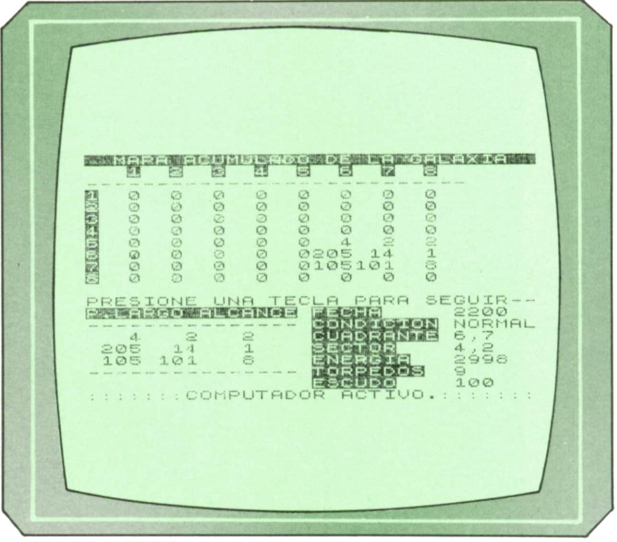



Fig. 3. Mapa acumulado de la galaxia.

Tú eres el capitán de la nave aventura. Dicha nave está dotada de motores termonucleares, de torpedos galácticos y de rayos láser en cantidad suficiente como para terminar la misión. En caso de que necesites más energía o más torpedos, puedes repostar en las bases que hay en la galaxia.

Tiene varios tipos de pantalla:

— La de corto alcance. Te dice exactamente dónde estás, cuántas estrellas y cuántos enemigos hay.

— La de largo alcance. Te muestra, mediante números, lo mismo que la de corto alcance, pero de más sectores.

— Mapa acumulado de la galaxia. Según te vas moviendo por la galaxia, el ordenador va almacenando toda la información que va apareciendo en la pantalla de largo alcance.

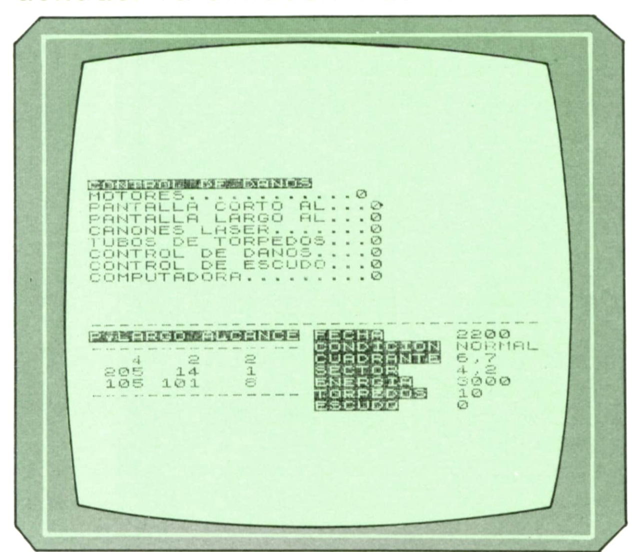




Fig. 4. Control de daños de la nave.

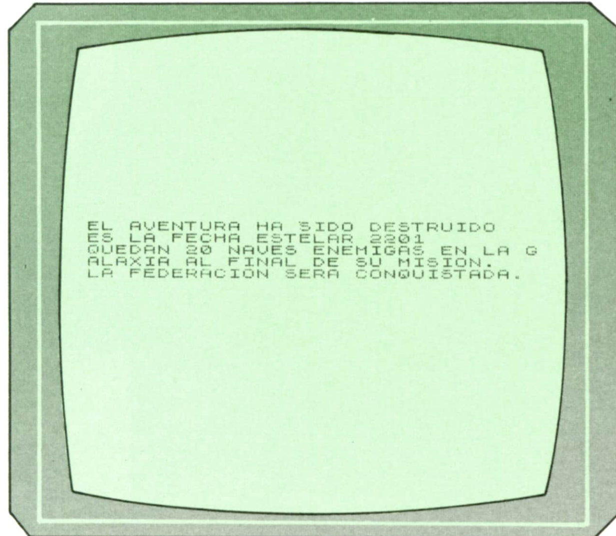
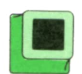




Fig. 5. Fin de la aventura.



Programa: Integrales para AMSTRAD por el método de Sympson

Este programa, muy parecido al que ya apareció en versión para IBM y compatibles, nos va a permitir calcular el valor de cualquier integral definida entre dos límites por el método de Sympson.

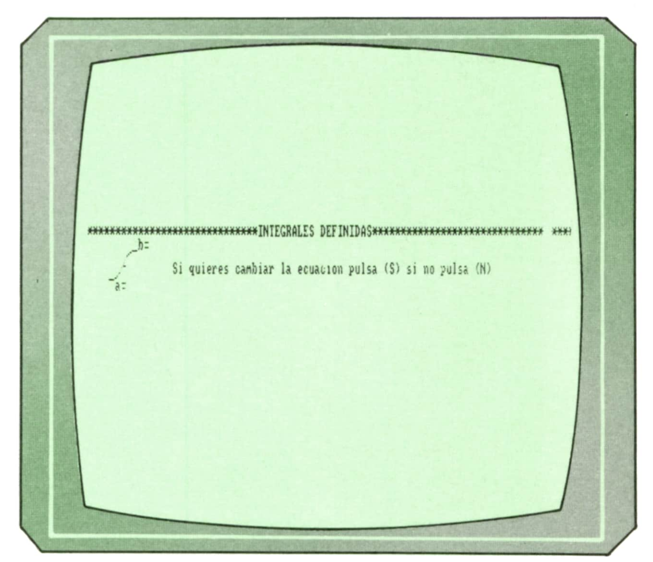




Fig. 6.

No hace falta explicar nada sobre el programa ya que éste es autoexplicativo y muy sencillo de usar.

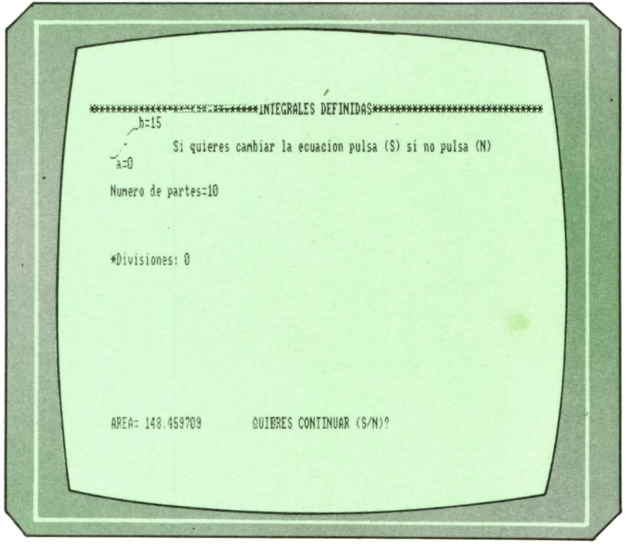




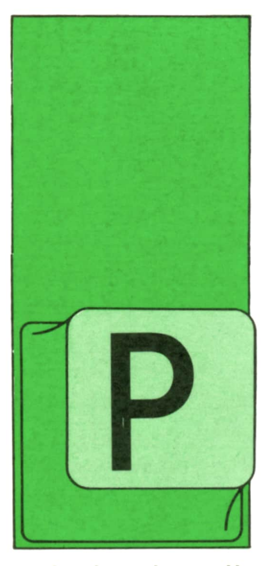
Fig. 7. Resultados del cálculo.

```
10 REM ******************************
20 REM ***  INTEGRALES SYMPSON.   ***
30 REM *** Un programa realizado ***
40 REM ***          Por          ***
50 REM *** Carlos A. Maria Morin ***
60 REM ***                       ***
70 REM ***     (C) Ediciones     ***
80 REM ***  Siglo Cultural 1987  ***
90 REM ******************************
100 REM
110 REM **************************************
120 REM *** PRESENTACION E INICIALIZACION ***
130 REM **************************************
140 REM
150 MODE 0
160 PAPER 0
170 PEN 2
180 BORDER 0
190 CLS
200 LOCATE 1,13
210 PRINT"I N T E G R A L E S"
220 FOR ret=1 TO 3000
230 NEXT
240 MODE 2
250 PEN 5
260 KEY 156,CHR$(13)+"goto 330"+CHR$(13)
270 KEY DEF 15,0,48,156
280 REM
290 REM ************************************
300 REM ******** PROGRAMA PRINCIPAL ********
310 REM ************************************
320 REM
330 CLS
340 LOCATE 1,1
350 PRINT"******************************INTEGRALES DEFINIDAS********************
**********"
360 GOSUB 1050:'DIBUJA SIMBOLO DE LA INTEGRAL
370 LOCATE 10,2
380 PRINT"b="
390 LOCATE 6,5
400 PRINT"a="
410 TAG
420 MOVE 120,355
430 PRINT"Si quieres cambiar la ecuacion pulsa (S) si no pulsa (N)";
440 TAGOFF
450 w$=INKEY$
460 IF w$="S" OR w$="s".THEN GOTO 920
470 IF w$="N" OR w$="n" THEN 490
480 GOTO 450
490 LOCATE 12,2
500 INPUT "",ls
510 LOCATE 8,5
520 INPUT "",li
530 LOCATE 5,7
540 PRINT"Numero de partes=";
550 INPUT "",pa
560 IF pa<1 THEN 530
570 REM
580 REM **************************************
590 REM ** SE INICIA EL PROCESO POR PARTES **
600 REM **************************************
610 REM
620 dx=(ls-li)/2/pa
630 to=0:x=li:GOSUB 820
640 to=to+y:x=x+dx:GOSUB 820
650 to=y*4+to:x=x+dx:GOSUB 820
660 to=y+to:pa=pa-1
670 LOCATE 5,12
```

```
680 PRINT"*Divisiones:";pa
690 IF pa<>0 GOTO 640
700 ka=to*dx/3
710 REM
720 REM *************************************
730 REM ****** SE VISUALIZA RESULTADO *******
740 REM *************************************
750 REM
760 FOR k=1 TO 300 STEP 10
770 SOUND 1,k,1,15
780 NEXT
790 LOCATE 5,24
800 PRINT"AREA=";ka;TAB(30);"QUIERES CONTINUAR (S/N)?"
810 GOTO 890
820 y=SQR(12^2-(x-12)^2)
830 RETURN
840 REM
850 REM *************************************
860 REM *** EDITA ECUACION Y PROCESA DATO ***
870 REM *************************************
880 REM
890 W$=UPPER$(INKEY$):IF W$="S" THEN 330
900 IF W$="N" THEN CLS:DELETE -1160
910 GOTO 890
920 TAG
930 MOVE 120,355
940 PRINT"
950 TAGOFF
960 LOCATE 32,25
970 PRINT"PULSA  [SHIFT+0]";
980 LOCATE 10,4
990 EDIT 820
1000 REM
1010 REM ************************************
1020 REM ************* DIBUJO ***************
1030 REM ************************************
1040 REM
1050 ORIGIN 70,351
1060 DEG
1070 MOVE 0,19
1080 FOR an=0 TO -90 STEP -20
1090 DRAW 19*SIN(an),19*COS(an)
1100 NEXT
1110 ORIGIN 31,352:MOVE 0,-19
1120 FOR an=0 TO -90 STEP -20
1130 DRAW -19*SIN(an),-19*COS(an)
1140 NEXT
1150 ORIGIN 0,0
1160 RETURN
```

TECNICAS DE ANALISIS

COMPROBACION DE PROGRAMAS (II)



ARA asegurar la corrección del proceso hay que establecer lo que se llama un «juego de ensayo»: conjunto de datos de entrada que someten al programa a todas las situaciones previstas, para comprobar la bondad del proceso realizado, en todos los casos posibles.



Preparación de los datos

A. *Entrada de los datos*

En principio la realización de estos juegos no debería ser complicada; sin embargo, hay casos en que la entrada de un proceso es la salida de otro, sin etapas intermedias y de tal modo que hay que modificar el segundo programa para que acepte los datos con el formato o el soporte en que nos es posible prepararlos: en estos casos hay que extremar los cuidados, para no modificar las condiciones en que posteriormente trabajará el programa, lo que, consecuentemente, invalidaría la prueba.

B. *Contenido del juego de ensayo*

Hay que examinar el modo en que realiza el programa su proceso y la disposición habitual de los datos de entrada para establecer el contenido del juego de ensayo: en efecto, si el conjunto de datos a procesar adopta una disposición fija y el proceso es uniforme, una *muestra* de los datos previstos es suficiente para comprobar que el tratamiento se realiza correctamente. En ocasiones, se prevé que pueda surgir algún tipo de dificultad con el volumen de información: en ese caso habrá que preparar un *juego de ensayo completo.*

Por otro lado, han de examinarse todas las circunstancias normales de proceso para prever que en cada juego aparezca al menos *un caso de cada tipo* básico. Además habrá de incluirse en cada batería de datos al menos *una aparición de cada caso atípico.*

Suele ser útil incluir, por último, algunos *casos adicionales aleatorios,* especialmente si el volumen de datos preparado de acuerdo con los criterios anteriores no es muy grande. Para realizar la generación de los números aleatorios con los que se van a construir los juegos de ensayo, se pueden utilizar diversos métodos: combinar de diferentes maneras los números de la sucesión de Fibonacci 1,1,2,3,5,8,13,21,34,55,89,144,... (obtenidos a partir de la definición $F(x+2) = F(x+1)+F(x)$ y de los valores iniciales $x0=1$ y $x1=1$), crear una sucesión de Von Neumann (tomando los números centrales de la sucesión de los cuadrados obtenidos a partir de una «simiente» elegida al azar: por ejemplo, para tener números aleatorios de tres cifras se parte de un número cualquiera, 378, y se va tomando sucesivamente 288 —el cuadrado de 378 es 142.884—, 583 —el cuadrado de 142.884 es 20.415.839.556— etc.), utilizar los sucesivos restos de las divisiones de dos números (es clásico el procedimiento de tomar dos números cualesquiera grandes, X e Y, e ir haciendo $S(X+1)=\text{resto}(X*S(X/Y))$, o cualquiera de los procedimientos que ofrecen los lenguajes de programación.

Debe tenerse en cuenta, por fin, el comentario hecho sobre las dificultades que, en ocasiones, surgen con el volumen de información a procesar, para producir un juego de datos de ensayo suficientemente grande, si este aspecto es de interés.

C. *Generación de los datos de ensayo*

El procedimiento más usual de obtener información sobre el contenido que deben tener los datos de prueba de un pro-

grama suele ser el simple *examen de los datos de entrada previstos;* en efecto, es usual limitar dicho conjunto de datos de ensayo a una reproducción de todos los posibles casos que se darán en la ejecución normal del programa.

Aunque en multitud de casos este procedimiento es suficiente, hay ocasiones en que la complejidad de los datos o de los procesos no asegura la comprobación de todos los casos posibles por este método; se suelen elaborar los datos en ese caso *a partir de las tablas de decisión* que definen el proceso a realizar por el módulo correspondiente.

Hay que reseñar, por último, que existen programas facilitados por las empresas de ordenadores o fabricantes de software, para la *generación automática de juegos de ensayo* de los programas. Estos programas generadores (usualmente ligados a un determinado lenguaje de programación) producen de un modo automático los datos de ensayo, a partir de las especificaciones que se les faciliten, según el formato requerido.

Obviamente si se dispone de un programa de estas características y los datos a generar lo permiten, este es el procedimiento más cómodo y más seguro de generar un juego de ensayo.



Comprobación de resultados

Normalmente, el proceso que debe realizar el programa es suficientemente conocido y sencillo como para *prever los resultados* que se deben obtener a partir de los datos de entrada; en consecuencia, la comprobación de la corrección del proceso se limita a evaluar los resultados esperados a partir de los datos del juego que se propone y compararlos con los que realmente se obtienen de la ejecución del programa. Es usual en este caso preparar una relación de los datos que forman parte del juego de ensayo, para anotar junto a ellos el resultado previsible y, por último, los resultados reales obtenidos; en la figura se incluye un diseño típico de esta clase de relaciones.

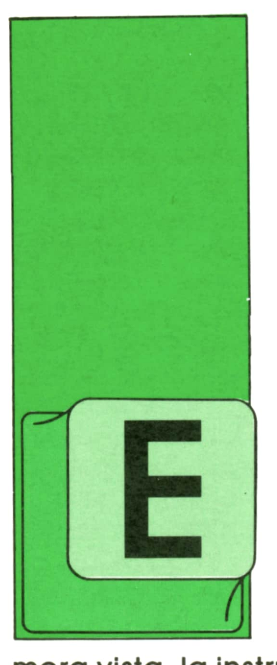
Hay ocasiones, sin embargo, en que dicha evaluación manual es sumamente complicada: en estos casos se suele introducir alguna o algunas *instrucciones de «seguimiento»* del proceso. Con estas instrucciones (que imprime algún comentario o el contenido de algún campo significativo cada vez que se ejecutan; es decir, cada vez que el programa pasa por ese punto) se va obteniendo una «traza» del proceso que se está realizando y se puede verificar la corrección del programa.

COMPROBACION DE PROGRAMAS

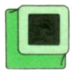
PROGRAMA:
CADENA: **HOJA**

	JUEGOS DE ENSAYO		RESULTADOS ESPERABLES		RESULTADOS OBTENIDOS	
	DATOS	VALORES	CAMPOS	VALORES	VALORES	OBSERV.
JUEGO N.º						
	DATOS	VALORES	CAMPOS	VALORES	VALORES	OBSERV.
JUEGO N.º						

DISEÑADO POR: FECHA:	CALCULADO POR: FECHA:	EJECUTADO POR: FECHA:



TECNICAS DE PROGRAMACION



Manejo de la pantalla

EN el capítulo anterior hemos visto cómo se puede escribir un texto, o el valor de una variable, o el resultado de una expresión, en la pantalla del ordenador. Pero, a primera vista, la instrucción PRINT (y sus equivalentes) es muy poco flexible, pues no permite elegir el lugar de la pantalla donde nos interesa que aparezca el texto. En efecto, cada instrucción PRINT escribe siempre el dato de que se trate en la primera línea disponible de la pantalla, cualquiera que ésta sea. En este capítulo vamos a ver cómo podemos alcanzar un dominio más completo de la pantalla, eligiendo a nuestra conveniencia el punto donde deseamos que se escriban nuestros textos y datos.

En el lenguaje BASIC existen dos instrucciones fundamentales cuya utilización nos permitirá alcanzar el dominio deseado. La primera instrucción es:

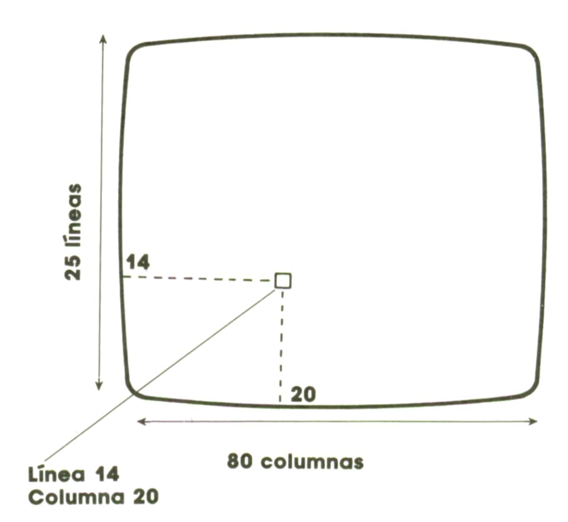
```
CLS
```

que borra completamente la pantalla, dejándola a nuestra disposición para que la llenemos con la información que deseemos.

Antes de explicar la segunda instrucción vamos a ver cómo se localiza una posición determinada en la pantalla de un ordenador.

Se considera que la pantalla está formada por un conjunto de líneas (por ejemplo, 25), en cada una de las cuales puede escribirse cierto número de caracteres (normalmente 40 u 80). Es decir, la pantalla puede considerarse como una disposición rectangular de lugares, o sitios, cada uno de los cuales puede ocuparse con un carácter o letra determinado. Supongamos que tenemos, en nuestro ordenador, una pantalla de 25 líneas de 80 caracteres. Esto significa que el número total de caracteres (o letras) que caben en la pantalla es igual a 25 × 80 = 2.000. Dicho de otro modo, en esa pantalla hay 2.000 lugares diferentes.

Cada uno de estos lugares queda definido por la posición que ocupa dentro de la pantalla de la siguiente manera: daremos el número de la línea y el número del lugar dentro de la línea. Por ejemplo: un lugar puede ser el número 20 de la línea 14. Decimos entonces que se encuentra en la línea 14 y en la columna 20 y lo representamos como en la figura:



Pues bien: la segunda instrucción BASIC de control de la pantalla tiene la siguiente forma:

```
LOCATE f,c
```

donde *f* y *c* son dos números (*f* debe estar comprendido entre 1 y el número de líneas de la pantalla, mientras *c* debe estar entre 1 y el número de caracteres por línea o columnas) y su efecto es colocar el cursor en la fila *f* y la columna *c* de la pantalla. Esto significa lo siguiente:

— Que, si la siguiente instrucción de lectura o escritura es INPUT, los caracteres que escribamos en el teclado irán

apareciendo en la pantalla a partir del lugar seleccionado por la instrucción LOCATE.

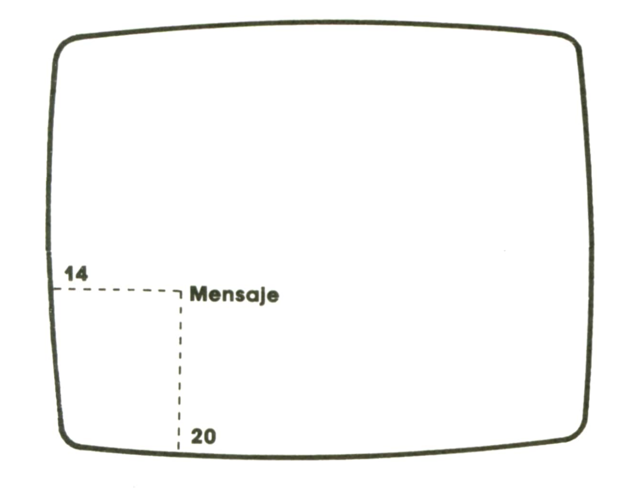
— Que, si la siguiente instrucción de lectura o escritura es PRINT, el texto o datos a escribir aparecerán a partir del lugar seleccionado por la instrucción LOCATE.

Veamos un ejemplo:



```
10 CLS
20 LOCATE 14,20
30 PRINT "Mensaje"
```

cuyo efecto es el siguiente (ver figura).



Definición de los campos de una pantalla

Ahora sabemos cómo puede situarse el cursor en un punto cualquiera de la pantalla, con objeto de que los mensajes y valores que escribamos aparezcan donde nos interesa. Vamos a ver ahora cómo podemos planificar el manejo de la pantalla, para facilitar la escritura de los programas.

Supongamos que queremos programar nuestra propia versión de un juego típico de guerra espacial, de los que existen numerosas variantes en el mercado. En este juego, el jugador deberá localizar y destruir, antes de que transcurra cierto tiempo, una flota de naves enemigas. La galaxia se divide en 64 cuadrantes, dispuestos en forma de tablero de ajedrez de 8 filas y 8 columnas. Además, cada cuadrante se divide también como un tablero de ajedrez en 8 por 8 sectores.

La sala de control de nuestra nave espacial contiene los siguientes elementos:

— Una pantalla de radar de corto alcance, que presenta una imagen del cuadrante en que se encuentra la nave.

— Una pantalla de largo alcance que muestra la situación en los cuadrantes fronterizos con el nuestro en forma de tabla de datos codificados de 3 filas y 3 columnas.

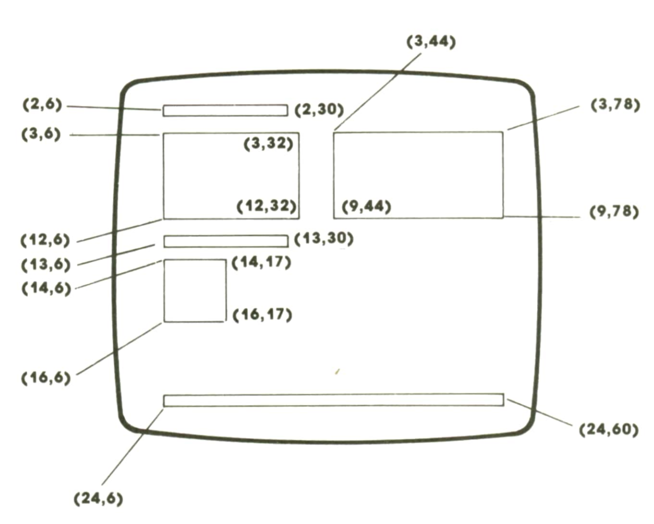
— Un informe condensado y permanente de la situación actual, que contiene información como la fecha, condición de la nave (normal, alerta amarilla o alerta roja), cuadrante en que nos encontramos, sector donde se encuentra situada la nave dentro del cuadrante, energía de que se dispone, número de torpedos a nuestra disposición y energía del escudo protector.

Todos estos datos estarán permanentemente visibles en la pantalla. Utilizándolos, el jugador debe decidirse por una de las acciones siguientes:

1. Navegación (trasladar la nave de sitio).
2. Disparar los cañones láser contra uno o varios enemigos.
3. Disparar un torpedo contra una nave enemiga.
4. Cambiar la energía asignada al escudo de protección.
5. Obtener información del control de daños.
6. Pedir información adicional a la computadora de a bordo.
7. Dar por terminado el juego.

En este tipo de juego es conveniente diseñar una estructura adecuada para la pantalla. Esta estructura podrá ser única o variable, según las circunstancias, pero en este último caso conviene que todas las posibilidades estén claramente definidas para facilitar la programación y el paso de una a la otra.

La figura siguiente muestra una posible estructura de la pantalla principal para nuestro ejemplo del juego espacial:



Observando la figura, podemos ver que hemos dividido la pantalla en seis campos bien definidos (conocemos las coordenadas fila-columna de todos sus vértices) entre los que pueden quedar espacios en blanco más o menos grandes. Estos campos son los siguientes:

1. Un campo rectangular de 10 filas y 27 columnas, donde aparecerá la pantalla de radar de corto alcance, que nos muestra la situación en el cuadrante en que nos encontramos. La primera fila y la última del campo servirán para escribir dos líneas continuas que sirvan de marco a la pantalla de radar. Cada una de las otras ocho líneas corresponderá a un sector dentro del cuadrante.

2. Un campo formado por una sola fila (la 2) y una longitud de 25 columnas. Aquí podremos situar un título para el campo anterior, tal como «Pantalla de corto alcance».

3. Un campo rectangular de 3 filas y 12 columnas, donde aparecerá la pantalla de radar de largo alcance, que nos muestra la situación en los cuadrantes vecinos. La información sobre el cuadrante en que nos encontramos estará situada en el centro.

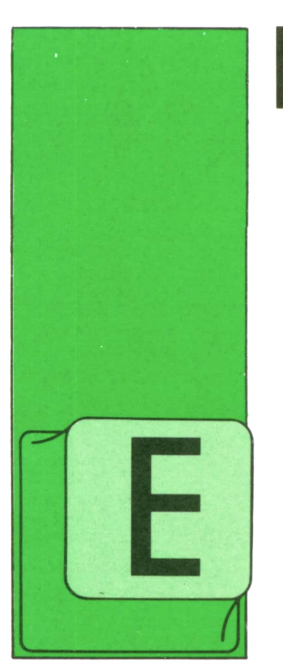
4. Un campo formado por una sola fila (la 13) y una longitud de 25 columnas. Aquí podremos situar un título para el campo anterior, tal como «Pantalla de largo alcance».

5. Un campo rectangular de 7 filas y 35 columnas, donde aparecerá el informe condensado de la situación de la nave. En este caso no necesitaremos título.

6. Un campo formado por una sola fila (la 24) y una longitud de 55 columnas. Aquí podremos situar un recordatorio de las acciones que podemos realizar y de qué tecla de función debemos presionar para conseguirlo.

En el próximo capítulo explicaremos cómo podemos escribir en cada campo la información deseada y cómo podemos conseguir que el programa sepa cuál de las teclas de función ha sido presionada para realizar la acción correspondiente.

LOGO



Operadores de relación

STE tipo de operadores nos sirve para comparar el valor de dos cosas. Son los siguientes:

= (igual que)
$<$ (menor que)
$>$ (mayor que)

Si ponemos

n1 = n2

o

p1 = p2

o

l1 = l2

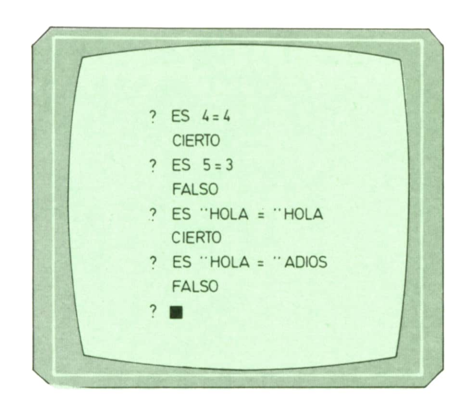
o

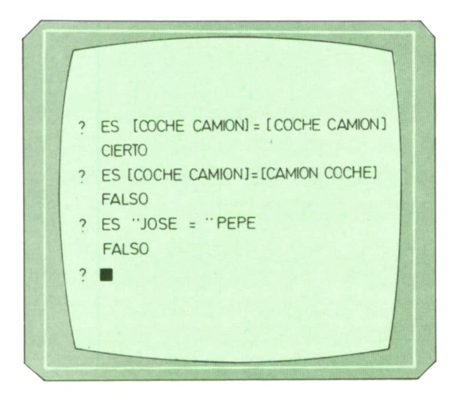
:v1 = :v2

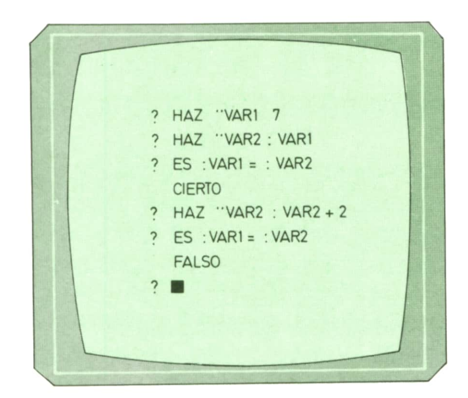
la tortuga devuelve CIERTO si

— n1 y n2 son números iguales
— p1 y p2 son palabras idénticas
— l1 y l2 son la misma lista
— v1 y v2 son variables que contienen el mismo valor

y FALSO en caso de ser diferentes.
Por ejemplo:







En cambio, si escribimos

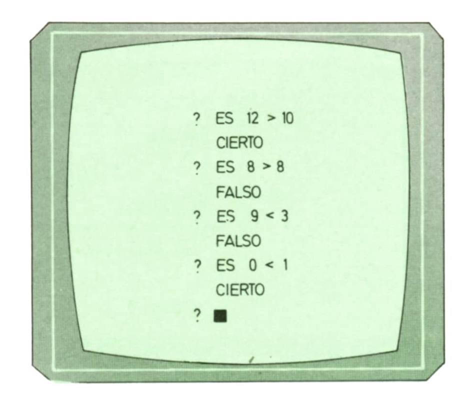
$n1 > n2$

la tortuga devuelve CIERTO si n1 es un número mayor que n2 y FALSO si n1 es menor o igual que n2, mientras que si ponemos

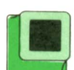
$n1 < n2$

el resultado será CIERTO si n1 es un número menor que n2 y FALSO si n1 es mayor o igual que n2.

Por ejemplo:



Los dos últimos operadores (> y <) también se pueden utilizar para comparar el contenido de dos variables cuyos valores sean números.



Operadores lógicos

Los operadores lógicos no sirven para formar condiciones compuestas, es decir, no se aplican a números, palabras, listas o variables, sino a condiciones que se han escrito usando los operadores anteriores.

El operador

NO cond

devuelve CIERTO si cond es falsa y FALSO si cond es cierta.

El operador

0 cond1 cond2

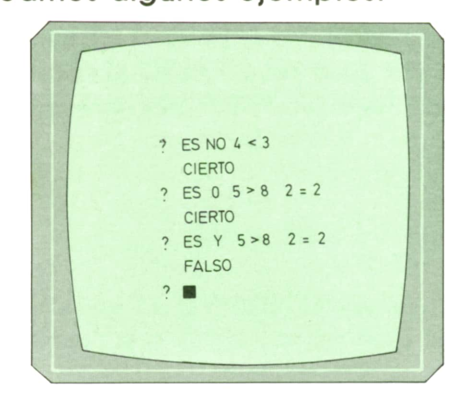
devuelve FALSO si cond1 y cond2 son falsas y CIERTO en caso contrario (sólo cond1 cierta, sólo cond2 cierta, ambas ciertas).

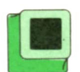
Por último, el operador

Y cond1 cond2

devuelve CIERTO si cond1 y cond2 son ciertas a la vez y FALSO si cualquiera de las dos o ambas son falsas.

Veamos algunos ejemplos:

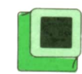




Os proponemos

1. Averigua el resultado que devolvería la tortuga al darle las siguientes condiciones:

```
?ES 1000 > 4                                →
? ES 12 < 8                                 →
? ES 4 = 4                                  →
? ES "M = "N                                →
? ES (A B C) = (C B A)                      →
? ES NO (1 2 3) = (1 3 2)                   →
? ES Y 4 < 10 4 > 0                         →
? ES 0 6 > 12 6 > 3 →
? HAZ "VAR1 0...............................→
? HAZ "VAR2 :VAR1 + 5
? ES :VAR1 > :VAR2                          →
? HAZ "VAR3 :VAR1
? ES 0 :VAR1 = :VAR2 :VAR1 = :VAR3          →
? HAZ "VAR1 :VAR2 + :VAR3
? ES NO :VAR1 < :VAR2                       →
```



El comando SI

Una vez que ya sabemos que una determinada condición es cierta o falsa necesitamos hacer algo con este resultado. Lo más normal es que mandemos a la tortuga ejecutar una serie de órdenes si esa condición es verdad.

Para eso tenemos que utilizar el comando

SI cond (lista)

donde cond es la condición y lista es el conjunto de órdenes que la tortuga tiene que ejecutar si el resultado de la condición anterior es CIERTO. En caso de que sea FALSO, estas órdenes no se realizan y la tortuga pasa a hacer las órdenes que estén detrás del comando SI.

Puede haber casos en que nos interese hacer cosas diferentes en función de que una condición sea cierta o falsa. Para ello podemos usar el comando SI de la siguiente forma:

SI cond (lista1) (lista2)

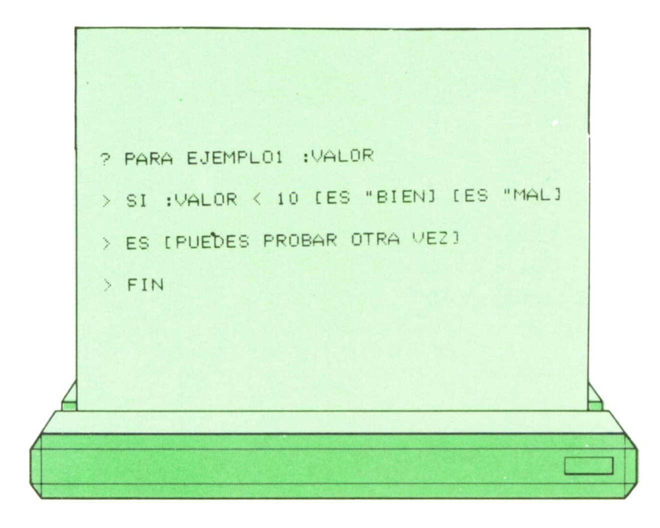
de manera que si el resultado de la condición cond es CIERTO, la tortuga ejecuta las órdenes de la lista 1 y si es FALSO las de la lista2.

Como es lógico, dentro de lista, lista1 y lista2 podemos poner cualquier comando válido, incluso otro SI o el nombre de procedimientos.

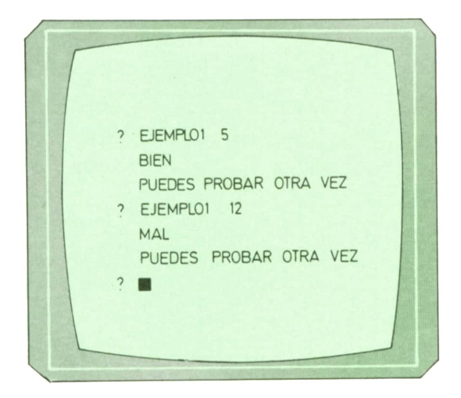
Vamos a ver la diferencia entre utilizar una forma u otra con un ejemplo. Supongamos que queremos escribir un proce-

dimiento que nos diga si un número es menor o no que 10.

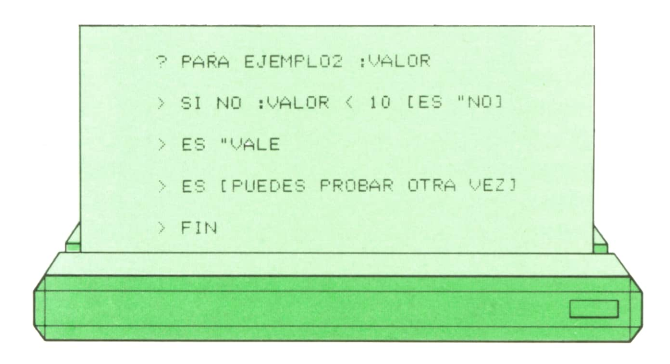
Utilizando la segunda forma nos queda:



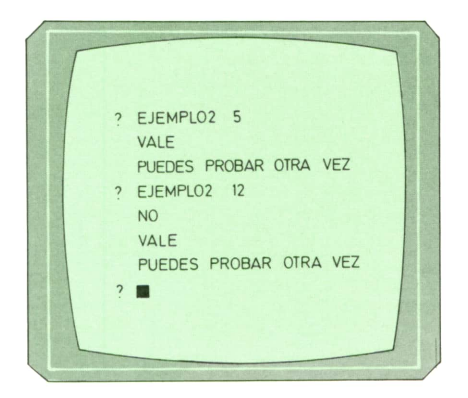
Si ahora lo ejecutamos nos queda:



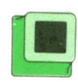
Usando la primera forma, el procedimiento podría ser:



y al ejecutarlo saldría:



Es decir, es importante darse cuenta de que la tortuga ejecuta las órdenes que vengan detrás del SI con independencia del resultado de la condición y del tipo de forma que utilizamos.



Utilización de SI en procedimientos recursivos

Hemos visto que en el caso de mandar a la tortuga ejecutar un procedimiento recursivo, ésta no para de hacerlo hasta que pulsemos la tecla de parada. Pues bien, existe otra manera de decírselo mediante la utilización del comando SI. De esta forma, le diremos a la tortuga que deje de realizar el procedimiento cuando se cumpla una determinada condición.

Para ello, tenemos que usar el comando



ALTO

dentro del procedimiento para detenerlo cuando se esté ejecutando.

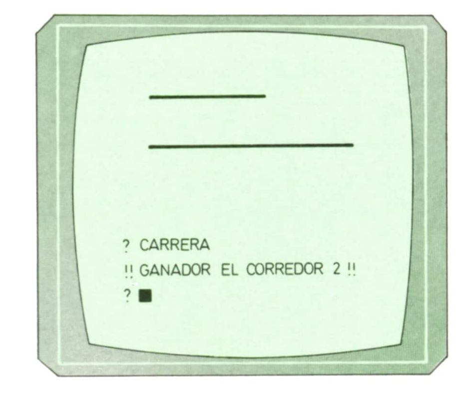
Por ejemplo, vamos a hacer una carrera con dos participantes. Para saber por dónde van pintaremos el rastro de cada uno en un color y daremos como ganador al primero que recorra 200 pasos desde la salida.

Necesitamos dos procedimientos: uno que coloque a los corredores en la línea de salida y otro recursivo que nos sirva para hacer la carrera. Los procedimientos serían:

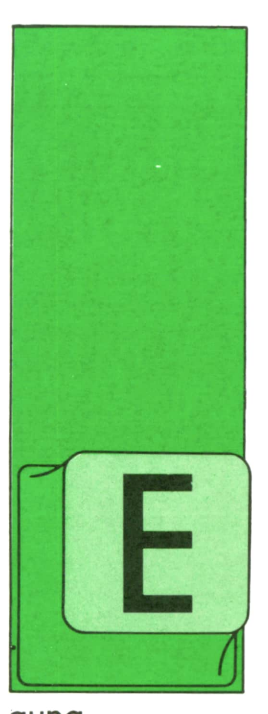
```
? PARA CARRERA
> BP
> HAZ "POS1 [-100 30]
> HAZ "POS2 [-100 -30]
> HAZ "NUMPASOS1 0
> HAZ "NUMPASOS2 0
> OT
> GD 90
> CORRER
> FIN
? PARA CORRER
>  SI  O  :NUMPASOS1  = 200 :NUMPASOS2 =  200  [SI  :NUMPASOS1  =
:NUMPASOS2  [ES  ";¡EMPATE!!]  [SI :NUMPASOS1  >  :NUMPASOS2  [ES
[;¡GANADOR  EL CORREDOR 1!!]] [ES [;¡GANADOR EL  CORREDOR  2!!]]]
ALTO]
> SL PONPOS :POS1 BL
> HAZ "PASOS AZAR 10
> PONCL 2
> AV :PASOS
> HAZ "NUMPASOS1 :NUMPASOS1 + :PASOS
> HAZ "POS1 POS
> SL PONPOS :POS2 BL
> HAZ "PASOS AZAR 10
> PONCL 8
> AV :PASOS
> HAZ "NUMPASOS2 :NUMPASOS2 + :PASOS
> HAZ "POS2 POS
> CORRER
> FIN
```

donde POS1 y POS2 son las posiciones de los dos corredores, NUMPASOS1 y NUMPASOS2 los pasos que ha dado cada uno y PASOS el número de pasos que han de avanzar.

Al ejecutarlos nos podría quedar lo siguiente:



PASCAL



Control del almacenamiento dinámico

EN todos los ejemplos con listas y árboles que hemos realizado hasta el momento se podían crear, en principio, tantos elementos como se necesitasen, sin limitación alguna.

Si, al intentar crear una nueva variable, no quedase ya suficiente espacio libre en la memoria del ordenador, se pararía el programa o bien se crearía una situación de «cuelgue» del ordenador, dependiendo del compilador empleado.

Por tanto, cuando existe la más mínima sospecha de que el espacio de memoria libre se puede agotar, cada vez que se vayan a crear nuevas variables se debe comprobar antes si todavía hay suficiente espacio para ellas.

La mayoría de los compiladores tienen alguna función para esto, que en algunos casos se denomina MEMAVAIL (contracción de Memory Available, memoria disponible en inglés) y que devuelve la cantidad de memoria libre (en bytes o en algún otro tipo de medida) en el momento de la llamada:

```
if MemAvail < Minimo then
  writeln ('Se acabó la memoria.');
```

La utilización del almacenamiento dinámico elimina la obligación de hacer previsiones sobre el número de datos que se van a procesar y permite, además, crear con ayuda de punteros estructuras de tipo lista o árbol. Para completar su utilidad sólo hace falta tener la posibilidad de, opcionalmente, dejar libre la zona de memoria de las variables dinámicas que no se vayan a utilizar más para así poder crear otras nuevas.

Esto se consigue en PASCAL mediante los procedimientos predefinidos MARK y RELEASE. Si escribimos:

```
Mark (P);
```

donde P puede ser un puntero de cualquier tipo, al ejecutarse el procedimiento se guarda en P la posición donde comienza la memoria disponible en ese momento. Posteriormente, si se ejecuta la instrucción:

```
Release (P);
```

toda la memoria que se reservó para variables dinámicas tras ejecutarse Mark (P) queda libre para otras nuevas. Los datos de aquellas variables se pierden y, por tanto, hay que tener mucho cuidado al utilizar estos procedimientos (dependiendo de en qué orden se hubiese construido un árbol, podrían quedar nodos desconectados de la raíz y serían, por tanto, irrecuperables).

Un ejemplo de utilización podría ser, en el programa de análisis de léxico que realizamos recientemente, ejecutar Mark (Punt) antes de empezar a crear el árbol de búsqueda y, una vez extraída la información, ejecutar Release (Punt) para volver a tener toda la memoria disponible. Punt, como ya se ha dicho, puede ser un puntero cualquiera y, por tanto, se podría definir así:

```
var Punt: ^integer;
```

o sea, un puntero apto para apuntar a variables de tipo integer.

Los compiladores más perfeccionados disponen también del procedimiento DISPOSE. Con él es posible dejar libre sólo la zona que corresponda a una variable específica. Si ejecutamos:

```
Dispose (Punt);
```

la zona ocupada por la variable a que apunta Punt quedará libre; en otras palabras, es una especie de «anti-NEW». Utili-

zando este método la memoria se libera a trozos, por lo que es posible que al cabo del tiempo, aunque la cantidad de memoria disponible sea grande, esté muy fragmentada.

Por ello suele haber, además de MemAvail o su equivalente, otra función para conocer el tamaño del mayor pedazo disponible, pues aunque hubiese suficiente memoria libre para una variable, pudiera ocurrir que no hubiese ningún pedazo lo suficientemente amplio para albergarla. Un nombre típico es MAXAVAIL.

En cualquier caso, el sistema MARK / RELEASE no conviene utilizarlo conjuntamente con DISPOSE a la hora de liberar memoria.

Primer ejemplo

El procedimiento InsertarEn del programa *Lexico* quedaría así si se desease controlar el espacio de memoria dinámica disponible:

```
procedure InsertarEn (var P: Puntero_t);
 begin
 if P = nil then
     if MemAvail >= Minimo then   (* crear nueva ficha *)
        begin
         new (P);
         with P^ do
           begin
            Palabra   := Nueva;
            Contador  := 1;
            Izquierdo:= nil;
            Derecho   := nil
           end
        end
      else
        writeln (Nueva,' no archivada.')
  else
     with P^ do
       begin
        case Test (Nueva, Palabra) of
          Iguales: Contador:= Contador + 1;
          Antes   : InsertarEn (Izquierdo);
          Despues: InsertarEn (Derecho)
        end
       end
 end;
```

Minimo sería una constante igual al espacio de memoria necesario para cada nodo expresado en las mismas unidades empleadas por la función MemAvail.

Con el Turbo-Pascal y los ordenadores personales IBM y compatibles, esa unidad es el «párrafo», que equivale a dieciséis bytes; el espacio de memoria de los diferentes campos de cada nodo sería:

Palabra:	30 bytes	(tantos como caracteres)
Contador:	2 bytes	
Izquierdo:	4 bytes	
Derecho:	4 bytes	
total:	40 bytes	

Sin embargo, el Turbo-Pascal dispone de la función SizeOf («tamaño de») que, aplicada a una variable o tipo, devuelve su tamaño en bytes; con ella, la comprobación del espacio libre quedaría así:

```
if MemAvail * 16 >= SizeOF (Nodo_t) then...
```

Segundo ejemplo

Vamos a modificar ahora el programa *Lexico* para que, tras presentar la lista de palabras por orden alfabético, las presente por orden de frecuencias.

Para ello iremos recorriendo los nodos del árbol original y los insertaremos en un nuevo árbol de búsqueda en el que el criterio de ordenación será el del orden de frecuencias. El orden de recorrido del árbol original afectará al orden en que

aparezcan las palabras con idéntica frecuencia y, para que sea alfabético, utilizaremos el recorrido en orden central. Será necesaria una variable de tipo Puntero_t, a la que llamaremos *RaizNueva*, para acceder al nuevo árbol.

Para no agotar la memoria del ordenador, a medida que se vayan guardando las palabras en el nuevo árbol, se liberará el espacio de su antiguo nodo. El procedimiento de creación del nuevo árbol sería:

```
procedure Transfiere (Origen: Puntero_t);

(*-------------------------------------------------------*)
(* Recorre los nodos del árbol apuntado por Origen y    *)
(* los inserta en un árbol de búsqueda por frecuencias *)
(*-------------------------------------------------------*)
 (*--------------------------------------------*)
 procedure GuardaEn (var P: Puntero_t);

   (* guarda el contenido de Origen^ en el árbol *)
   (* apuntado por P según el valor del contador *)

  begin
  if P = nil then
      begin
       new(P);
       with P^ do
         begin
          Palabra   := Origen^.Palabra;
          Contador  := Origen^.Contador;
          Izquierdo:= nil;
          Derecho   := nil
         end
      end
    else
      with P^ do
        if Origen^.Contador > Contador then GuardaEn (Izquierdo)
                                       else GuardaEn (Derecho)
 end;
(*--------------------------------------------*)
begin
 if Origen <> nil then with Origen^ do
   begin
    Transfiere (Izquierdo);
    GuardaEn (RaizNueva);
    Transfiere (Derecho);

    (* tras salvar el contenido del nodo y sus descendentes: *)
    Dispose (Origen)
   end
end;
```

En la parte principal del programa, tras la presentación de la lista por orden alfabético, pondríamos:

```
RaizNueva:= nil;
Transfiere (Raiz);
writeln;
writeln ('Lista por orden de frecuencias:'); writeln;
Diferentes:= 0; TotalPalabras:= 0;
Muestra_Arbol (RaizNueva);
writeln;
writeln ('Palabras: ',Diferentes,'   Total: ',TotalPalabras);
```

El procedimiento *GuardaEn*, cuando llega a un nodo con contador inferior al del que queremos guardar, avanza por el subárbol izquierdo y, cuando es superior, por el derecho; cuando los dos contadores son iguales avanza también por el derecho, por lo que el nuevo nodo quedará por detrás según el nuevo criterio de ordenación. Esta circunstancia, unida al recorrido en orden central del árbol original, garantiza la aparición por orden alfabético de las palabras con idéntico contador.

El procedimiento de transferencia puede ser mejorado desde el punto de vista de la velocidad si, con cada palabra, en lugar de crear un nuevo nodo, copiar los campos y borrar el anterior, utilizamos el mismo nodo retocando los punteros para que pase a pertenecer al nuevo árbol:

```
procedure Transfiere (Origen: Puntero_t);

(*-----------------------------------------------------------*)
(* Recorre los nodos del árbol apuntado por Origen y los *)
(* transfiere a un árbol de búsqueda por frecuencias      *)
(*-----------------------------------------------------------*)
  var SalvaDerecho: Puntero_t;
  (*-----------------------------------------------*)
 procedure GuardaEn (var P: Puntero_t);

  (* integra Origen^ en el árbol apuntado *)
  (* por P según el valor del contador    *)

  begin
  if P = nil then P:= Origen    (* P apunta ahora al mismo nodo *)
    else
      with P^ do
        if Origen^.Contador > Contador then GuardaEn (Izquierdo)
                                       else GuardaEn (Derecho)
  end;
 (*-----------------------------------------------*)
 begin
  if Origen <> nil then with Origen^ do
    begin
      Transfiere (Izquierdo);

      GuardaEn (RaizNueva);
      SalvaDerecho:= Derecho;
      Derecho      := nil;
      Izquierdo    := nil;

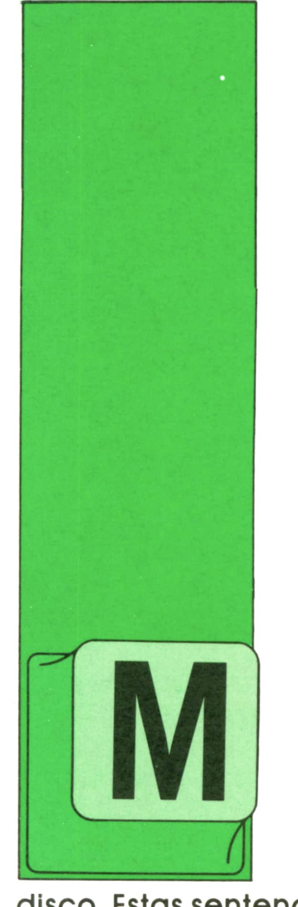
      Transfiere (SalvaDerecho)
    end
 end;
```

Nada más integrar un nodo en el nuevo árbol, se convierte en un nodo hoja, por lo que hay que anular sus punteros; sin embargo, como el recorrido del árbol original es en orden central, su puntero *Derecho* se necesita todavía; por otra parte, no se puede esperar a transferir su subárbol derecho para anularlos, pues quizá durante ese proceso haya que engancharle descendentes en el nuevo árbol.

Para resolver esto es para lo que se emplea la variable *SalvaDerecho* (también se podría sacar la copia de *Derecho* antes de llamar a *GuardaEn* y encargarle a éste la anulación de los punteros).

Al primer procedimiento *Transfiere* se le podría añadir control de memoria, pero al último, dado que no se crean nuevos nodos, ni tiene sentido ni resulta posible.

OTROS LENGUAJES





Entrada salida

MUCHOS lenguajes, como el PASCAL, tienen sentencias para leer y escribir datos de forma secuencial del teclado, de la pantalla del ordenador o de archivos en disco. Estas sentencias son muy prácticas por su facilidad de manejo, pero existen aplicaciones en que las propiedades de los recursos son muy infrautilizadas si son accedidas mediante las sentencias estándar del lenguaje. La generalización en estas sentencias conllevan que las soluciones más óptimas para algún tipo de recursos sean realmente ineficientes para otros. Así, pues, en los lenguajes de programación más modernos se tiende a hacer transparentes al usuario las propiedades de algunos recursos para aplicaciones que requieran un uso muy eficiente de éstos.

En MODULA-2 este problema se ha resuelto sin incluir ninguna sentencia de entrada salida como parte integrante del lenguaje. Esta solución ha sido aceptada debido a dos facilidades del MODULA: la existencia de módulos permitiendo la construcción jerarquizada de módulos de bibliotecas representando niveles crecientes de abstracción; y las facilidades de bajo nivel que permiten acceder directamente a los sistemas de entrada salida.

Si un programador desea ignorar los detalles concretos del manejo de los diversos recursos sólo debe utilizar los módulos con un mayor nivel en la jerarquía dentro de la biblioteca. Si, por el contrario, desea hacer un uso muy eficiente de ellos, utilizará los de menor nivel. En este último caso hay que tener en cuenta que esto causaría que el programa no se podría transportar fácilmente a otro ordenador diferente o bajo otro sistema operativo.

Ahora explicaremos cómo está organizado el módulo estándar InOut donde se encuentran estas funciones, refiriéndonos sólo a las de mayor nivel. Debemos recordar que no es parte integrante del lenguaje, sino sólo unas bibliotecas que están accesibles en todos los compiladores de MODULA.

Dentro de los datos distinguiremos entre datos legibles e ilegibles. Los legibles son los dedicados a la comunicación con seres humanos, como los escritos en la pantalla o los leídos del teclado (suelen ser de tipo CHAR, excepto si se trata de gráficos); los ilegibles son los que maneja el ordenador internamente y los que utiliza para comunicarse con sus periféricos, o con otros ordenadores, pudiendo ser de cualquier tipo de datos.

En cualquiera de los dos casos las operaciones suelen efectuarse de forma secuencial.

Antes de presentar los datos, las funciones deben ser capaces de realizar algunas operaciones con ellos para que sean legibles. Esta transformación se conoce como formatear los datos, por ejemplo, cuando se escribe un número éste debe convertirse en una secuencia de caracteres.

Para generalizar los dispositivos a utilizar se considerarán los 'stream', que son objetos con las siguientes características:

— Todos los elementos del 'stream' son del mismo tipo, que será conocido como tipo base del 'stream'. Si este tipo es CHAR, entonces será conocido como 'text stream'.

— El número de elementos del 'stream' no es conocido *a priori*, por lo que será una estructura dinámica de datos.

— El stream sólo puede modificarse añadiendo elementos al final, o borrando toda la estructura.

— Sólo un elemento del 'stream' es accesible a un mismo tiempo, y éste es conocido como la posición dentro del 'stream'.

— Todo 'stream' tiene un modo de acceso: lectura o escritura.

Esta es la máxima generalización posible de un fichero de entrada salida. En PASCAL un 'stream' sería algo parecido a una FILE.

Veamos los procedimientos para leer datos de un 'stream'; éstos son:

Read (Ch)	lee un carácter.
ReadString (s)	lee una tira de caracteres.
ReadInt (x)	lee un número entero.
ReadCard (x)	lee un número de tipo CARDINAL.

El final del 'stream' es reconocido mediante la variable importada Done.

```
Read (ch)
WHILE Done DO
  Procesar (ch);
  Read (ch);
END
```

Su valor es FALSE si en la operación de lectura no ha tenido éxito debido al final del 'stream'.

Los procedimientos para escribir datos son:

Write (C)	escribe un carácter.
WriteString (s)	escribe una tira de caracteres.
WriteLn	escribe un carácter de fin de línea.
WriteInt (i,n)	escribe un entero con *n* caracteres.
WriteCard (c,n)	escribe un número con *n* caracteres.
WriteOct (x,n)	escribe un número en octal.
WriteHex (x,n)	escribe un número en hexadecimal.

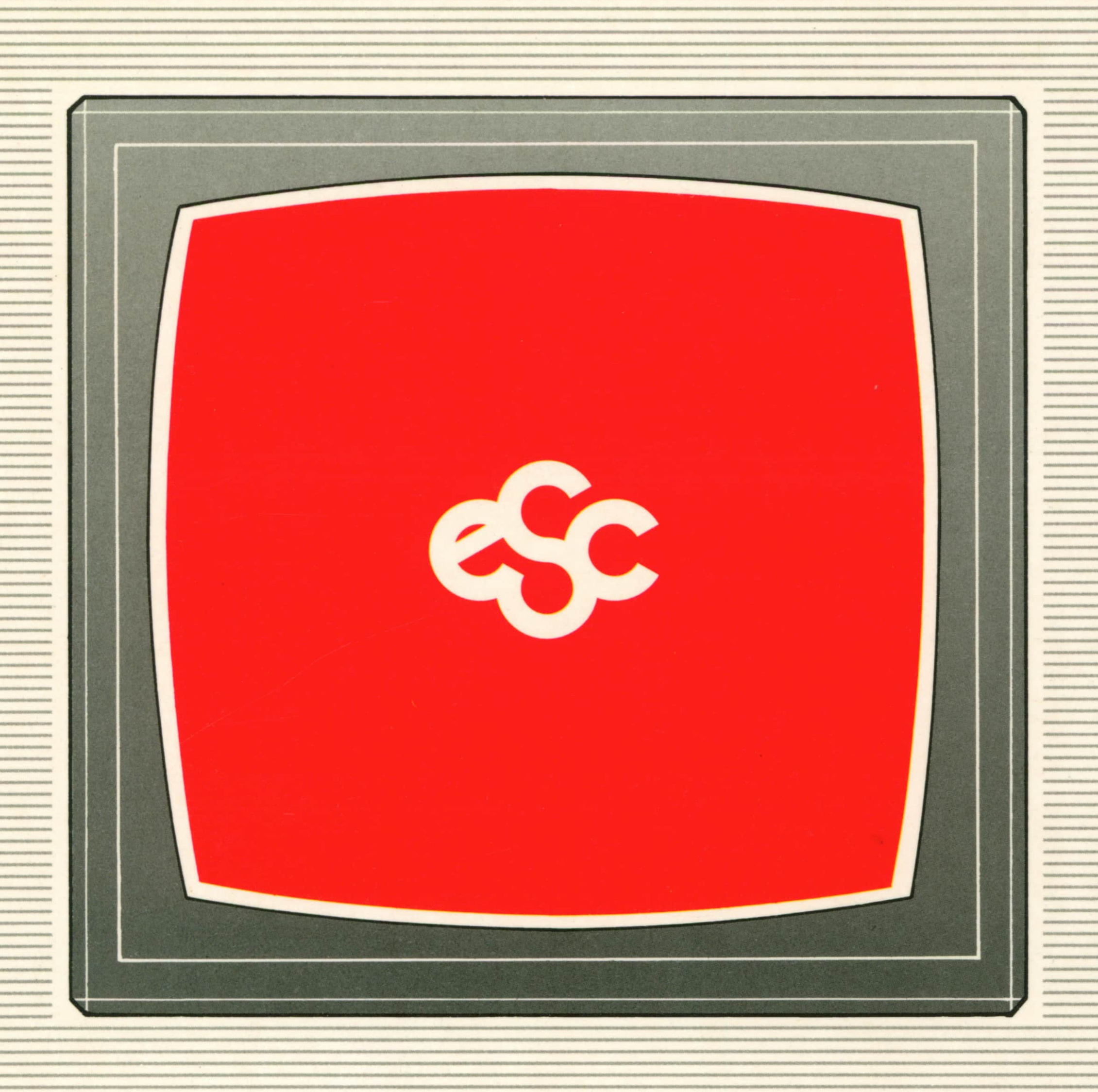
Para el manejo de los 'steam' existen las siguientes funciones:

OpenInput (s)	prepara el 'stream' para entrada.
OpenOutput (s)	prepara el 'stream' para salida.
CloseInput	cierra el 'stream'.
CloseOutput	cierra el 'stream'.

Estas funciones sirven para asociar ficheros con los 'stream' estándar In y Out. A menos que se declare explícitamente, se toma como entrada el teclado y como salida la pantalla. Debe tenerse la precaución de cerrar todos los ficheros antes de acabar el programa, o nos arriesgamos a graves problemas, como perder el fichero, e incluso el disco sobre el que lo grabamos.

Para la lectura y escritura de reales existe el módulo estándar RealInOut.

Quien desee más información sobre los módulos de entrada salida de menor nivel le indicamos que consulte el manual de su compilador.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼